



**COMPUTER VISION TRACKING USING PARTICLE FILTERS FOR 3D
POSITION ESTIMATION**

THESIS

Kyle D. Kenerley, Second Lieutenant, USAF

AFIT-ENY-14-M-28

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENY-14-M-28

COMPUTER VISION TRACKING USING PARTICLE FILTERS FOR 3D POSITION
ESTIMATION

THESIS

Presented to the Faculty
Department of Aeronautical and Astronautical Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Astronautical Engineering

Kyle D. Kenerley, B.S.A.E.
Second Lieutenant, USAF

March 2014

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

COMPUTER VISION TRACKING USING PARTICLE FILTERS FOR 3D POSITION
ESTIMATION

Kyle D. Kenerley, B.S.A.E.
Second Lieutenant, USAF

Approved:

//signed//
Alan L. Jennings, PhD (Chairman)

17 Mar 2013
Date

//signed//
Jonathan T. Black, PhD (Member)

11 Mar 2014
Date

//signed//
Daniel D. Doyle, PhD (Member)

11 Mar 2014
Date

Abstract

This line of research seeks to increase knowledge of a tracked target using the particle filter, also known as Sequential Monte Carlo (SMC) methods. The target is tracked using vision based observations. These observations were simulated using both dual cameras and a single camera. If only a single camera tracks the target, depth cannot be determined directly and is considered an unobservable state. Filters can estimate this unobservable state using a dynamic model and data from the image. However the movement of the target is nonlinear which eliminated filters traditionally used to track motion such as the Kalman filter and its variants. The particle filter is an alternative that can track nonlinear motion, but was not feasible until recently due to its computational requirements. Simulations of nonlinear target movement, first in two dimensions, then three, evaluated the particle filter's feasibility and performance. Subsequent simulations evaluated the particle filter's ability to track a target using dual and single camera observations. Evaluation tests were devised to characterize the performance of each filter. Analysis metrics were produced to analyze the results of these tests. Linear and Kalman filters were also devised to serve as additional comparisons to the particle filter. Results for dual camera observations demonstrated the filter could track the target and determine unobservable states, however results for the single camera observations indicated the filter was problematic since it could not return accurate depth estimates and suffered from severe weight collapse.

Table of Contents

	Page
Abstract	iv
Table of Contents	v
List of Figures	ix
List of Tables	xii
List of Symbols	xiv
List of Acronyms	xviii
 I. Introduction	 1
1.1 General Issue	1
1.2 Problem Statement	1
1.3 Research Objectives/Questions/Hypothesis	2
1.4 Research Focus	3
1.5 Methodology	3
1.6 Assumptions/Limitations	3
1.7 Implications	4
1.8 Overview	4
 II. Literature Review	 5
2.1 Overview	5
2.2 Photogrammetry	5
2.3 Optical Flow	5
2.3.1 Limitations of Existing System	7
2.4 Filters	7
2.4.1 Bayes' Theorem	7
2.4.2 Bayes Filter	8
2.4.3 Kalman Filter	10
2.4.3.1 Kalman Filter Prediction Step	10
2.4.3.2 Kalman Filter Update Step	11
2.4.4 Extended Kalman Filter	11
2.5 Particle Filter	12
2.5.1 Goal of the Particle Filter	12

	Page
2.5.2 Monte Carlo Methods	14
2.5.3 Importance Sampling	15
2.5.4 Sequential Importance Sampling	16
2.5.5 Particle Filter Process	17
2.5.5.1 Initialization	17
2.5.5.2 Importance Sampling	17
2.5.5.3 Selection	18
2.5.5.4 Re-sampling Algorithm	19
2.5.5.5 Visual Depiction of Particle Filter Process	20
2.5.6 Limitations of the Particle Filter	21
2.5.7 Uses of the Particle Filter	22
2.6 Conclusion	22
III. Methodology: Prototype Filters	23
3.1 Introduction	23
3.2 System Design Approach	23
3.3 Prototype Particle Filter A: Single Variable Tracking	23
3.4 Prototype Particle Filter B: Multi-Variable Tracking and Hidden State Determination	25
3.4.1 Target Model	25
3.4.2 Measurements	26
3.4.3 Particle Filter Model	26
3.4.4 Kalman Filter Model	28
3.4.5 Simulation Scenarios	29
3.4.5.1 Scenario 1: No System or Measurement Noise	30
3.4.5.2 Scenario 2: Measurement Noise, No System Noise	34
3.4.5.3 Scenario 3: System Noise and Measurement Noise	37
3.4.5.4 Scenario 4: Weight Discrepancy and Collapse	41
IV. Methodology: Application Filters	47
4.1 Introduction	47
4.2 System Design Approach	47
4.2.1 Coordinate Frames	47
4.2.2 Target Model	51
4.2.3 Measurement Model	52
4.2.4 Global Frame Equations	53
4.2.5 Camera Frame Equations	55
4.2.6 Pixel Frame Equations	55
4.2.7 Acceleration and Depth Knowledge	58
4.3 Evaluated Particle Filter A	58

	Page
4.3.1 Initialization and Proposal Distribution	58
4.3.2 Importance Sampling	59
4.3.3 Resampling and Global State Estimation	60
4.3.4 Preliminary Evaluation	60
4.4 Evaluated Particle Filter B	61
4.4.1 Preliminary Evaluation	65
4.5 Weight Collapse Mitigation	66
4.5.1 Depth Vector Variation	67
4.5.2 State Jacobian Variation	67
4.5.3 Weighing Matrix Adjustment	69
V. Simulation Tests and Results	70
5.1 Introduction	70
5.2 Particle Filter Performance Evaluation	70
5.2.1 Performance Metrics	71
5.2.1.1 Mean Absolute Error	71
5.2.1.2 Threshold Error Range	72
5.2.1.3 Metric Variables	73
5.2.2 Comparison Models	74
5.2.2.1 Linear Model A: Filter A	74
5.2.2.2 Linear Model B: Filter B	75
5.2.2.3 Evaluate Kalman Filter A	76
5.3 Evaluated Particle Filter A	78
5.3.1 Orthogonal Axes Movement	79
5.3.2 Circular Movement	99
5.3.3 Unconstrained Motion	118
5.3.4 Evaluated Particle Filter A Summary	125
5.4 Evaluated Particle Filter B	125
5.4.1 Equal Matrix	128
5.4.2 Unequal Matrix	134
5.4.3 Evaluated Particle Filter B Summary	146
VI. Conclusions and Future Work	147
6.1 Conclusion	147
6.2 Practical Considerations	148
6.3 Future Work	151
Appendix: MATLAB Code	153
A.1 Introduction	153

	Page
A.2 Prototype Particle Filter A	153
A.3 Prototype Particle Filter B	156
A.4 Evaluated Particle Filter A	159
A.4.1 EPF_A_Execute_Final	159
A.4.2 EPF_A_Function_Final	186
A.5 Evaluated Particle Filter B	204
A.5.1 EPF_B_Execute_Final	204
A.5.2 EPF_B_Function_Final	236
Bibliography	267

List of Figures

Figure	Page
2.1 Visual Depiction of the Discrete Probability Distribution	13
2.2 The State-Space	14
2.3 Bootstrap Particle Filter Process [11, 12]	20
3.1 Prototype Particle Filter A: Non-linear Function Demonstration	24
3.2 Prototype Particle Filter B (PPF-B): Mean Error for No System or Measure- ment Noise	31
3.3 PPF-B: Single Run for No System or Measurement Noise	32
3.4 PPF-B: Mean Error for No System Noise	35
3.5 PPF-B: Single Run for No System Noise	36
3.6 PPF-B: Mean Error for System and Measurement Noise	39
3.7 PPF-B: Single Run for System and Measurement Noise	40
3.8 PPF-B: Weight Discrepancy and Collapse	43
3.9 PPF-B: Iteration Prior to Weight Collapse	45
4.1 Global Transformation from MATLAB Default Orientation	48
4.2 Transformation from Global to Camera Frame	49
4.3 Transformation from Camera to Pixel Frame	50
4.4 Target in Global Frame	51
4.5 Target Centroid and Measurement Points	53
4.6 Relationship Between Focal Length and View Angle	57
4.7 Filter A Preliminary Evaluation: Position	62
4.8 Filter A Preliminary Evaluation: Velocity	63
4.9 Particle Filter B Weight Collapse	66
5.1 Threshold Error	73

Figure	Page
5.2 Performance Metric Variables	74
5.3 Mean Error for Axial Movement without Measurement Noise	82
5.4 Threshold Error for Axial Movement without Measurement Noise	83
5.5 Single Simulation for x-Axis Movement without Measurement Noise	84
5.6 Single Simulation for y-Axis Movement without Measurement Noise	85
5.7 Single Simulation for z-Axis Movement without Measurement Noise	86
5.8 Comparison of Threshold Values for X-Axis Movement with No Measurement Noise	90
5.9 Mean Error for Axial Movement with Measurement Noise	92
5.10 Threshold Error for Axial Movement with Measurement Noise	93
5.11 Single Simulation for x-Axis Movement with Measurement Noise	94
5.12 Single Simulation for y-Axis Movement with Measurement Noise	95
5.13 Single Simulation for z-Axis Movement with Measurement Noise	96
5.14 Mean Error for Circular Rotation without Measurement Noise	101
5.15 Threshold Error for Circular Rotation without Measurement Noise	102
5.16 Single Simulation for xy-Plane Circular Rotation with Measurement Noise . . .	103
5.17 Single Simulation for xz-Plane Circular Rotation with Measurement Noise . . .	104
5.18 Single Simulation for yz-Plane Circular Rotation with Measurement Noise . . .	105
5.19 Target and EPF-A Variable Relationships	107
5.20 Mean Error for Circular Rotation with Measurement Noise	111
5.21 Threshold Error for Circular Rotation with Measurement Noise	112
5.22 Single Simulation for xy-Plane Circular Rotation with Measurement Noise . . .	113
5.23 Single Simulation for xz-Plane Circular Rotation with Measurement Noise . . .	114
5.24 Single Simulation for yz-Plane Circular Rotation with Measurement Noise . . .	115

Figure	Page
5.25 Single Simulation for Unconstrained States, in Global Frame, with Measure- ment Noise	119
5.26 Mean Error for Unconstrained Motion	121
5.27 Threshold Error for Unconstrained Motion	122
5.28 MAE for Equal Weight Matrices with System Noise	124
5.29 MAE for Equal Weight Matrices with No Noise	129
5.30 TER for Equal Weight Matrices with No Noise	130
5.31 MAE for Equal Weight Matrices with Noise	132
5.32 TER for Equal Weight Matrices with Noise	133
5.33 MAE for Unequal Weight Matrices without Noise	135
5.34 TER for Unequal Weight Matrices without Noise	136
5.35 MAE for Unequal Weight Matrices with Noise	138
5.36 TER for Unequal Weight Matrices with Noise	139
5.37 Mean and Threshold Depth Error	141

List of Tables

Table	Page
3.1 PPF-B Scenario: Initial Conditions	29
3.2 PPF-B Scenario: Parameters	30
3.3 Mean Errors for Last 50 Time Steps without System or Measurement Noise . .	33
3.4 PPF-B Scenario 2: Variances	34
3.5 Mean Errors without System or Measurement Noise	37
3.6 PPF-B Scenario 3: Variances	38
3.7 Mean Errors without System or Measurement Noise	41
3.8 PPF-B Scenario 2: Variances	41
4.1 Global Frame Target States	52
4.2 Image Properties	57
4.3 Filter A Preliminary Evaluation Initial Conditions	60
4.4 Filter A Preliminary Evaluation Measurement Noise Variances	61
4.5 Filter A Preliminary Evaluation: Mean Errors	61
4.6 Filter B Preliminary Evaluation Initial Conditions	65
5.1 EPF-A Scenario Parameters	79
5.2 Orthogonal Axes Initial Conditions	80
5.3 Measurement Noise Variations	80
5.4 Evaluated Particle Filter A Legend Acronyms	81
5.5 Mean Axial Errors for Last 50 Time Steps without Measurement Noise	88
5.6 Mean Axial Errors for Last 50 Time Steps with Measurement Noise	98
5.7 Circular Rotation Initial Conditions	100
5.8 Circular Motion Time Unit Ranges for Mean Error	108
5.9 Mean Circular Errors for Variable Time Steps without Measurement Noise . . .	109

Table	Page
5.10 Mean Circular Errors for Variable Time Steps with Measurement Noise	117
5.11 Unconstrained Motion Initial Conditions	118
5.12 Mean Unconstrained Motion Errors for Last 50 Time Steps	123
5.13 Mean Unconstrained Motion Errors with System Noise for Last 50 Time Steps	125
5.14 EPF-B Scenario Initial Conditions	127
5.15 EPF-B Scenario Parameters	127
5.16 Measurement Noise Variations	128
5.17 Measurement Noise Variations	128
5.18 EPF-B Time Unit Ranges for Mean Error	142
5.19 EPF-B and Comparison Mean Errors for Measurements without Noise	143
5.20 EPF-B and Comparison Mean Errors for Measurements with Noise	144
5.21 Weight Collapse for Measurements without and with Noise	145

List of Symbols

Symbol	Definition
X, Y, Z	global principal axes
U, V, W	camera principal axes
U_p, V_p	pixel principal axes
x, y, z	position in global frame
u, v, w	position in camera or pixel frame, depending on subscript
V	speed/velocity magnitude (component of velocity)
θ	tilt (component of velocity)
ϕ	pan (component of velocity)
a	acceleration
$\dot{\theta}$	tilt angle rate of change
$\dot{\phi}$	pan angle rate of change
α	pan angle (deg) of camera
β	tilt angle (deg) of camera
\mathbf{x}	states
\mathbf{y}	measurements
\mathbb{H}	hypothesis
\mathbb{D}	data
$p()$	discrete probability distribution (unless otherwise noted)
π	proposal distribution
ω	importance weight
i	index of normalized weights
δ	Dirac-delta function
\mathbf{x}	state vector

Symbol	Definition
u	process input
y	measurement vector
w	state (system) noise
v	measurement noise
<i>g</i>	state propagation vector
<i>h</i>	measurement propagation vector
A	state transition model matrix
B	control input model
u_k	Kalman control vector
Q	state (system) noise covariance
C	observation model
K	Kalman gain
I	identity matrix
R	measurement noise covariance
<i>n</i>	individual particle (sample)
<i>N</i>	total number of particles (samples)
$\sigma^2()$	variance
$\sigma()$	standard deviation
<i>r_u</i>	uniform random number
<i>r_n</i>	normal random number
<i>N_{eff}</i>	number of effective particles
Ω	weighing matrix
Ω()	component of weighing matrix
<i>m</i>	individual observed point
<i>M</i>	total number of observed points

Symbol	Definition
fl	focal length
γ	angular error between filter and target
ΔX	change in hidden states
ΔS	change in observation states
s_{f_1}, s_{f_2}	scaling factors 1 and 2
λ	eigenvalues
V	eigenvectors
\mathbf{x}_{MAE}	state mean absolute error
\mathbf{x}_S	vector of states sorted low to high
\mathbf{x}_{TER}	state threshold error range
τ	percentile value for TER
q	number of simulations
D	Euclidean distance between filter prediction and target
R_{DCM}	rotation direction cosine matrix

Subscripts

G	global frame
C	camera frame
p	pixel frame
t	discrete time step
T	target
f	filter
LM	linear model
k	Kalman filter

Symbol	Definition
--------	------------

Superscripts

\sim	unweighted states
\wedge	normalized weights
T	transpose

List of Acronyms

Acronym	Definition
AFIT	Air Force Institute of Technology
ASIR	Auxiliary Sampling Importance Re-sampling
BPF	Bootstrap Particle Filter
DC	Depth-Compensated Evaluation Particle Filter B
DCM	Direction Cosine Matrix
DPD	Discrete Probability Distribution
EKF	Extended Kalman Filter
EW	Equally Weighted Weighing Matrix
PDF	Probability Density Function
EKF-A	Evaluation Kalman Filter A
EPF-A	Evaluation Particle Filter A
EPF-B	Evaluation Particle Filter B
IS	Importance Sampling
JC	Jacobian-Compensated Evaluation Particle Filter B
MAE	Mean Absolute Error
MC	Monte Carlo
NC	Non-Compensated Evaluation Particle Filter B
PKF-A	Prototype Kalman Filter A
PKF-B	Prototype Kalman Filter B
PMF	Probability Mass Function
PPF-A	Prototype Particle Filter A
PPF-B	Prototype Particle Filter B
PTZ	Pan-Tilt-Zoom

Acronym	Definition
RPF	Regularized Particle Filter
SIR	Sequential Importance Resampling
SIS	Sequential Importance Sampling
SLMA	Simple Linear Model A
SLMB	Simple Linear Model B
SMC	Sequential Monte Carlo
TER	Threshold Error Range
UW	Unequally Weighted Weighing Matrix
2-D	Two-Dimensional
3-D	Three-Dimensional

COMPUTER VISION TRACKING USING PARTICLE FILTERS FOR 3D POSITION ESTIMATION

I. Introduction

1.1 General Issue

With the advent of computers approximately 50 years ago, there is a general trend pushing computer systems to simulate natural abilities, such as vision and motion. What seems natural to humans, such as spotting and following a moving object such as a Frisbee, is challenging for computers. The two elements used by humans are the eyes to see and follow the target, and the brain to process the images. The analogous computer components are pan/tilt/zoom Pan-Tilt-Zoom (PTZ) cameras to 'see' and track the target and software to process the images from the cameras. Humans are innately able to differentiate between a moving target, other targets, the background, and any appearance of movement due to movement of the eyes or heads. Computers are unable to perform these differences without complex software. Several key challenges confront these artificial systems, among them, the ability of a system to identify moving targets and attempt to follow them by predicting their path against a moving background (i.e. simulating moving eyes and heads of humans). The focus of this thesis explores a potential method of predicting the movement of targets by using the particle filter. The particle filter attempts to determine the location and predict the velocity and heading of a target as much as humans do when catching a Frisbee.

1.2 Problem Statement

The research in this thesis attempts to answer two problems that currently face visual tracking systems, in particular the visual tracking system being developed at the Air Force

Institute of Technology (AFIT). The first problem is the ability of the PTZ cameras to track a target. Presently, the tracking algorithm looks for a change in a target's position and does not attempt to account for a target's changing velocity when moving the PTZ cameras to follow the target. The tracking algorithm does not account for a target's changing velocity because the only incoming data is how the target moved relative to its last position, or optical flow. The particle filter attempts to predict the target's 3-D position, velocity and heading estimates so the cameras can more accurately follow the target. The particle filter uses a non-linear model of a target's movement in 3-D space. The second problem is depth determination using a single camera. Typically two or more cameras are used to determine the depth or how far away a target is. A single camera is unable to make this determination since the camera does not know if the target is changing size or moving towards or away from the camera. The only data available is the target's 2-D optical flow; side-to-side and up-and-down movement, along with the velocities. The particle filter attempts to predict the depth based on the 2-D optical flow and the velocities as well as any change in the target's scale size.

1.3 Research Objectives/Questions/Hypothesis

This research focuses on characterizing the ability of the particle filter to determine the states of a target in 3-D space using two cameras or using a single camera. The sub-objectives that must be accomplished to fulfill this goal are detailed below.

- Develop a general model of the tracked object's dynamics
- Determine if the particle filter can determine hidden states
- Develop a particle filter using measurements from two cameras and evaluate its performance using simulated data
- Develop a particle filter using measurements from a single camera and evaluate its performance using simulated data

1.4 Research Focus

The focus of this research is to evaluate the viability of the particle filter to a). predict a target's position, velocity, and heading states and b). predict depth of a target using a single camera. The filter results will be compared with a known truth to provide a baseline comparison.

1.5 Methodology

Model development and simulated data collection will occur within the MATLAB environment. The path and states of the target moving through Three-Dimensional (3-D) space will be generated using a non-linear model. A collection of points for the simulated camera to track will be generated around the moving target point. The particle filter will use the simulated camera's measurements of these points as the measurement data and generate its predicted location of the target in 3-D space. Both the dual camera and single camera particle filter's will follow this procedure.

1.6 Assumptions/Limitations

The movement of a target in 3-D space is typically nonlinear, thus several assumptions concerning the model must be made in order to make the problem tractable theoretically. The model assumes no changes in acceleration rates of any states. The target is assumed to not change in size (i.e. the model does not change in form in any way). Thus any change in size detected by the camera correlates exclusively to a change in distance from the camera. These model assumptions limit the effectiveness of the filter, however a simpler model provides a basis for expansion into more complex models. Noise inherent in the model is assumed to be Gaussian white noise, that is noise with a normal distribution and zero mean. Additionally, the state and measurement variances must be known or approximated so that the filter can be tuned properly.

1.7 Implications

The PTZ system the particle filter could be implemented on uses two cameras to determine the location of the target and tracks based on position of the target. The particle filter has the potential to enable accurate velocity tracking using only position data. With accurate velocity estimates, better predictions of the target's future movement could be made. With better predictions, the PTZ cameras could be steered to more accurately to track the target resulting in better tracking. Additionally, the particle filter also has the potential to reduce the number of cameras to a single camera decreasing system complexity and cost.

1.8 Overview

The subsequent chapter is a review of the literature concerning the optical flow algorithm, and the motivation to select the particle filter. Additionally, Chapter 2 also details the methodology of the particle filter and how it works to return state estimates. Chapter 3 describes several simple particle filters, referred to as prototype particle filters Prototype Particle Filter A (PPF-A) and PPF-B. These prototype filters were developed to both demonstrate the particle filter as well as verify basic properties of the particle filter, notably its ability to determine hidden states. Chapter 4 describes the target model and particle filters used to simulate the PTZ system for both dual and single cameras. Note, these particle filters, Evaluation Particle Filter A (EPF-A) and Evaluation Particle Filter B (EPF-B), are different from the prototype filters PPF-A and PPF-B. Chapter 5 discusses the data analysis of EPF-A and EPF-B and extrapolates results from the data analysis. Chapter 6 elaborates on the conclusions drawn from the data analysis and provides recommendations for future research and development.

II. Literature Review

2.1 Overview

With the increase in computational speed and portability as well as high-definition digital optics, there has been a significant amount of research undertaken into the development of vision-based real-time tracking systems [1]. The current evaluation system used for this report has well developed tracking capabilities, using optical flow background estimation as its basis [10]. However it presently possesses a linear Kalman filter for target tracking, which is not optimized for non-linear target tracking. Chapter 2 summarizes the applicable research conducted concerning optical flow processing as it relates to the tracking system used along with a discussion on filters, with a focus on particle filters.

2.2 Photogrammetry

Photogrammetry is the process of determining 3-D coordinates through images. The mathematical underpinnings of photogrammetry are rooted in the 1480s with Leonardo da Vinci's study of perspectives [8, p. 1]. However, digital photogrammetry did not emerge until the development of the digital cameras in the 1970s. A close relative of photogrammetry, videogrammetry, determines information based on multiple images taken over time. Currently photogrammetry and videogrammetry are used in a variety of fields from topographic mapping to film motion capture and special effects [18, p. 4].

2.3 Optical Flow

Photogrammetry, as it pertains to this research, aims to determine the three-dimensional coordinates of the target's centroid based on multiple two-dimensional feature points seen by the camera. Features represent the object in an image, and a feature is required to determine the location of a point or set of points that represent an object; hence

the term feature points. Feature points are points that can be well described mathematically. Typical features and definitions are as follows: [12, p. 2-5]

- Color - the apparent color of an object is influenced by two physical factors: 1) spectral power distribution of the illuminant; 2) the surface reflectance properties of the object
- Edges - strong changes in image intensities defining object boundaries; less sensitive to illumination changes
- Optical Flow - dense field of displacement vectors defining the translation of brightness for each location in consecutive frames
- Texture - statistical measures of the intensity variation of a surface which quantifies the properties such as smoothness and regularity; less sensitive to illumination changes

While color is a popular feature, the tracking algorithm developed by Doyle uses optical flow to detect feature points [12]. The motivation behind this choice is that optical flow is able to track a wider range of targets than other methods that are more sensitive to a target's physical properties. Moving a camera results in a shift of the background image. All tracking algorithms must discern between actual target movement and apparent movement induced by the movement of the cameras. By using optical flow to track the target's movement, any additional movement of the background can also be tracked using optical flow and then subtracted from the target's apparent movement to reveal the target's actual movement. Another method to improve tracking is to train a system to recognize the target; however training is undesired in order to allow the tracking algorithm to function on a variety of targets and surfaces. Thus, none of the filters used target training.

2.3.1 Limitations of Existing System.

As typical with many photogrammetry systems, X and Y coordinates are relatively accurate, however Z (depth) coordinates are slightly less accurate [12]. Additionally, the current system requires two cameras for depth estimation and can only track one object at a time. Although the current system tracks position and velocity states using a Kalman filter, the weight for velocity is minimal due to the filter's uncertainty. This uncertainty stems from the simple control law used by the filter, to regulate the target to the center of the camera's vision field.

2.4 Filters

In order to produce meaningful information concerning the location of the tracked object in the global frame, the incoming pixel data must be filtered. Although a variety of methods exist, this research focused on Bayesian based filters for reasons detailed in the subsequent sections. The filter ultimately used, the particle filter, is explored in depth in Section 2.5.

2.4.1 Bayes' Theorem.

Bayes' Theorem is named for Thomas Bayes and was heavily edited and updated by Richard Price and published in 1763 [3]. The theorem evaluates the probability that a proposed hypothesis, or prediction, is correct given given observational, or measurement data, about that hypothesis.

$$P(\mathbb{H}|\mathbb{D}) = \frac{P(\mathbb{H}) \times P(\mathbb{D}|\mathbb{H})}{P(\mathbb{D})} \quad (2.1)$$

$P(\mathbb{H})$ is the prior or initial degree of belief in the hypothesis, or predicted states. $\frac{P(\mathbb{D}|\mathbb{H})}{P(\mathbb{D})}$ represents the probability, or how strongly the data \mathbb{D} , or measurements, support the hypothesis. The result is posterior probability, or degree of belief in the hypothesis having accounted for the data \mathbb{D} . Using multiple hypotheses with the same data produces a Probability Density Function (PDF). The PDF embodies all available statistical

information and may be described as the complete solution to an estimation problem [2, p. 174]. Specifically, the PDF describes the relative likelihood, probability, that a particular variable will be equal to a certain value. However, the Bayesian estimation is sensitive to noise that affects the measure of the hypothesis or states \mathbf{x} [6]. Thus due to the amount of noise present in the data or measurements of the pixels as well as in the camera PTZ movements, a filter is needed that can recover the actual state values from noisy measurements \mathbf{y} . The subsequent filters utilize the state-space model below:

$$\begin{aligned}\mathbf{x}_{t+1} &= g(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \\ \mathbf{y}_t &= h(\mathbf{x}_t, \mathbf{v}_t)\end{aligned}\tag{2.2}$$

The functions g and h describe state and measurement propagation. \mathbf{x} is the state, \mathbf{u} is the process input, \mathbf{y} is the measurement, \mathbf{w} and \mathbf{v} are the system (or state or process) and measurement noise vectors, and t is the discrete time [20].

2.4.2 Bayes Filter.

An extension of the Bayes' Theorem is Recursive Bayesian Estimation also known as the Bayes Filter or Grid-Based Filter. The variables, $\mathbf{x}_{0:t}$ and $\mathbf{y}_{1:t}$ are denoted in Equation 2.3 and 2.5 [11, p. 5].

$$\mathbf{x}_{0:t} \triangleq \{\mathbf{x}_0, \dots, \mathbf{x}_t\}\tag{2.3}$$

$$\mathbf{y}_{1:t} \triangleq \{\mathbf{y}_1, \dots, \mathbf{y}_t\}\tag{2.4}$$

These are the state values and measurements up to time-step t , respectively. Using a given system and measurement model, the Bayes Filter calculates a new posterior PDF, $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, as well as the marginal posterior PDF, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, at each time-step based on the previous state estimate and new incoming measurement. The marginal posterior is also known as the filtering distribution or posterior filtered density. The filtering distribution returns the probabilities of various values of variables, the states in this research, without reference to the values of other variables. This is the posterior PDF of the states \mathbf{x} at a

single time step, t . Ultimately, the filtering distribution is the PDF of interest since the state probabilities are separate and not joint. The posterior PDF is given by Bayes' Theorem at any time t , as seen in Equation 2.5 [2, p. 108].

$$p(\mathbf{x}_t|\mathbf{y}_{t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{t-1}) d\mathbf{x}_{t-1} \quad (2.5)$$

A recursive formula for the posterior PDF, $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, is derived and shown in Equation 2.6 [11, p. 6].

$$p(\mathbf{x}_{0:t+1}|\mathbf{y}_{1:t+1}) = p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}) p(\mathbf{x}_{t+1}|\mathbf{x}_t)}{p(\mathbf{y}_{t+1}|\mathbf{y}_{1:t})} \quad (2.6)$$

The filtering distribution can be calculated in a two step recursion process: the prediction and update, as seen in Equation 2.7 and 2.8 [11, p. 6].

$$\text{Prediction: } p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} \quad (2.7)$$

$$\text{Update: } p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{\int p(\mathbf{y}_t|\mathbf{y}) p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) d\mathbf{x}_t} \quad (2.8)$$

A prior PDF, or the prediction as seen in Equation 2.7, for the state \mathbf{x}_k is obtained using the Chapman-Kolmogorov equation based on the measurements up to $t - 1$ [21, p. 531]. The state is first predicted with the state propagation belief $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ using the model f . Then this prediction is corrected by the measurement likelihood $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$. The update follows Bayes' Theorem to calculate the posterior PDF for state \mathbf{x}_t , accounting for measurements up to time step t , $\mathbf{y}_{1:t}$. These expressions however, are deceptively simple. In reality, one cannot typically calculate the normalizing constant, $p(\mathbf{y}_t)$, and the marginals of the posterior, $p(\mathbf{x}_t|\mathbf{y}_t)$, since they require the evaluation of complex high-dimension integrals [11, p. 6]. Furthermore, the Bayes Filter scales poorly in reality due to the large state space needed for multidimensional state vectors. Multidimensional integrals are needed to calculate the prior probability of each point in the state space. These integrals also grow and become incalculable as the state space grows. Additionally, the state space must be discretized or certain limitations must be placed on the model for computer computation.

Much research has been devoted to developing approximations for these distributions. The subsequent sections discuss some of the common filters used. Each of the following filters attempts to approximate the posterior PDF

2.4.3 *Kalman Filter.*

The Kalman filter is a popular type of Bayes filter with several restrictions to solve the problem of state estimation encountered in the Bayes Filter [11] [20]:

- g and h must be linear functions
- w and v must be uncorrelated, additive Gaussian white noise, with zero mean and known variance

The Kalman filter is unsuitable to model a target moving in three dimensional space with changing velocity and acceleration. A model with changing velocity and acceleration is highly likely to be non-linear and the Kalman filter requires linear models. However, the Kalman filter serves as a useful comparison to the particle filter and a second order Kalman filter is used as a comparison within Chapters 4 and 5. The Kalman filter is broken into two steps, the prediction and update, like the Bayes filter it emulates.

2.4.3.1 *Kalman Filter Prediction Step.*

The prediction step consists of a prediction of the states, $\tilde{\mathbf{x}}_t$, and a prediction of the error covariance matrix, $\tilde{\mathbf{P}}_t$, which is an estimate of how accurate the state estimate is. Additional parameters are the state transition model matrix, \mathbf{A} , the control-input model, \mathbf{B} , the control vector, \mathbf{u}_k , and the system noise covariance matrix, \mathbf{Q} . Equation 2.9 and 2.10 describe the prediction step [4, p. 411-470] [7, p. 231-279].

$$\tilde{\mathbf{x}}_t = \mathbf{A}\tilde{\mathbf{x}}_{t-1} + \mathbf{B}\mathbf{u}_t \quad (2.9)$$

$$\tilde{\mathbf{P}}_t = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^\top + \mathbf{Q} \quad (2.10)$$

2.4.3.2 Kalman Filter Update Step.

The update step occurs once measurements are received by the filter. The update step calculates the Kalman gain, \mathbf{K}_t , which is a measure of the relative certainty of the state estimate and measurements at each time step. The predicted states updated with the Kalman gain the measurements. The final update is an update of the error covariance matrix using the Kalman gain. The measurements used are defined by the measurement model, \mathbf{C} ; some corresponding states may be observable and some may be hidden (no measurements exist for them). Equation 2.11, 2.12, and 2.13 describe the update process [4, p. 411-470] [7, p. 231-279].

$$\mathbf{K}_t = \tilde{\mathbf{P}}_t \mathbf{C}^\top (\mathbf{C} \tilde{\mathbf{P}}_t \mathbf{C}^\top + \mathbf{R})^{-1} \quad (2.11)$$

$$\mathbf{x}_t = \tilde{\mathbf{x}} + \mathbf{K}_t (\mathbf{y}_t - \mathbf{C} \tilde{\mathbf{x}}_t) \quad (2.12)$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}) \tilde{\mathbf{P}}_t \quad (2.13)$$

Two key weaknesses of the Kalman filter that are made clear are the inability to change the system and measurement noise covariance matrices, \mathbf{Q} and \mathbf{R} , and the inability to change its dynamics (\mathbf{A}). For a second-order Kalman filter, the control law defines the acceleration values; if the target acceleration is not constant, the Kalman filter will begin to lag. A vision tracking system possesses a significant problem since the PTZ cameras may no longer move fast enough to capture the target if the filter is used to control the cameras.

2.4.4 Extended Kalman Filter.

To combat the limitations of the Kalman filter, namely its requirement for linear functions, the Extended Kalman Filter (EKF) was developed at NASA Ames [19] [22]. The EKF approaches the linear problem by linearizing a process before estimating it similarly to a Kalman filter. The EKF calculates the Jacobian of f and h around the estimated state, thus producing a trajectory of the model function centered around the state in question [20]. Unfortunately it can still fail to accurately represent the model unless a number of specific parameters are applied rendering the filter unsuitable for situations requiring long

sequences or sudden changes in the state of the target [11, p. 5] [6, p. 3]. Furthermore, if the initial estimate is incorrect, the filter may quickly diverge due to its linearization around the estimate. Such a condition is unsuitable for tracking since initial estimates of the target's position and velocities are likely to be inaccurate. The original purpose of the EKF was to predict navigation properties for orbital flybys. The creators assumed that the spacecraft would not deviate far or quickly from its intended path and thus the process could be linearized around the estimate [19, p. 614-615].

2.5 Particle Filter

The origins of the particle filter, also known as Sequential Monte Carlo (SMC) methods, actually predate the Kalman filter by several years. The fundamental basis of the particle filter was established by Hammersley in 1954, wherefore he argued that SMC methods could be used to estimate the posterior PDF of the state-space by using Bayesian recursion equations [14]. However, it was not until 1993 that Gordon et al. demonstrated that SMC methods are suitable for tracking or other applications that previously relied on Kalman filters or their variants [13]. Particle filters were also not practical until sufficient computer power was available to run the filter in real time. The significant benefit of the particle filter is that no restrictions are placed on f_i or h_i , or on the distributions of the system or measurement noise [13, p. 108-109]. Furthermore, the particle filter is not dependent on knowledge of prior states, only the present state. This aspect is discussed further in Section 2.5.1. The prime difference between the particle filter and Bayes' or Grid Filters is that although both seek to determine the PDF, the particle filter does so discretely using a set of random samples, or particles, instead of as a function over the state space. As the number of particles increases, they provide a more accurate representation of the PDF.

2.5.1 Goal of the Particle Filter.

The goal of a particle filter is to estimate the posterior PDF of the state variables given the measurements. However, as discussed in Section 2.4.2, determining the actual

continuous PDF is typically impossible, and if not impossible, computationally expensive, slow, and prone to noise. Rather than determine the posterior PDF, the particle filter estimates the PDF discretely by generating a Discrete Probability Distribution (DPD). There are two elements to the DPD: a probability distribution and a Probability Mass Function (PMF). The probability distribution is a sample, $\mathbf{x}_t^{(n)}$, at time-step t of $n = 1, \dots, N$ samples drawn from the proposal distribution, N being the total number of samples drawn. These samples are also known as the particles. The proposal distribution is a DPD that estimates the desired posterior PDF and may also be referred to as the importance sampling distribution, the importance function, or the importance density, depending on the literature source, and is discussed further in Section 2.5.3. The PMF is referred to as the importance weights. Conceptually, the probability distribution is the spread of particles within the state-space before the weights are assigned. The importance weights assign the probabilities to this distribution. The weights represent the likelihood of a particle containing the truth states. Figure 2.1 illustrates the two elements of the DPD. Several

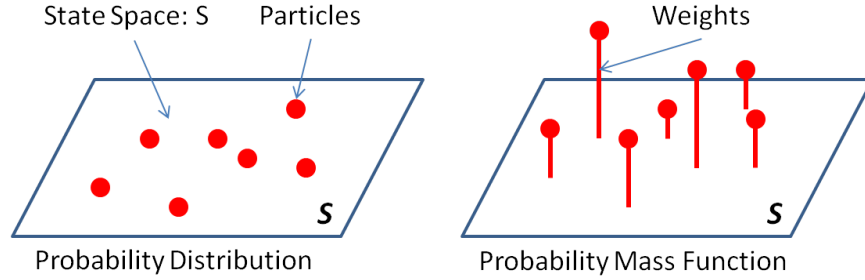


Figure 2.1: Visual Depiction of the Discrete Probability Distribution

assumptions regarding the model must be made before using the particle filter [11, p. 5].

- The state process, \mathbf{x}_t is a first order Markov process that can be modeled as:

$$\mathbf{x}_t | \mathbf{x}_{t-1} \sim p_{\mathbf{x}_t | \mathbf{x}_{t-1}}(\mathbf{x} | \mathbf{x}_{t-1})$$

with an initial DPD of $p(x_0)$. A Markov process is a process that satisfies the Markov property. A process satisfies the Markov property if predictions for future states can be made based only on present states just as well as if all previous states were known.

- Measurements \mathbf{y}_t are conditionally independent provided that \mathbf{x}_t is known; that is, each measurement \mathbf{y}_t is only dependent on its corresponding state \mathbf{x}_t

$$\mathbf{y}_t | \mathbf{x}_t \sim p_{\mathbf{y}|\mathbf{x}}(\mathbf{y} | \mathbf{x}_t)$$

A visual illustration of the state-space is seen in Figure 2.2.

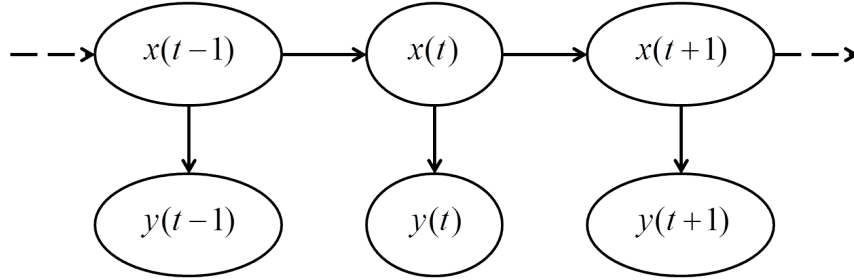


Figure 2.2: The State-Space

Section 2.5.2 through 2.5.4 detail the development of the elements used by the particle filter. The actual particle filter process is detailed in Section 2.5.5.

2.5.2 Monte Carlo Methods.

As mentioned in Section 2.5, the particle filter is based on Monte Carlo (MC) methods. MC methods seek to address the problems encountered with Bayes's Filter, detailed in Section 2.4.2. MC methods approximate the integrals present in Equation 2.6, 2.7, and 2.8 [11, p. 6]. However, MC methods usually cannot sample from the the posterior PDF $p(\mathbf{x}_{0:t} | \mathbf{y}_{1:t})$ at any time t [11, p. 8]. Thus, alternate methods were developed to address this issue.

2.5.3 Importance Sampling.

Importance Sampling (IS) is a technique to estimate the properties of a specific PDF using samples generated from a different PDF, assuming these PDFs are directly proportional to each other. Suppose $p(x) \propto \pi(x)$, where $p(x)$ is a PDF from which it is difficult or impossible to draw samples from, but $\pi(x)$ may be easily sampled from. Additionally, $p(x)$ may only be evaluated up to proportionality. Let these samples be defined as x^n sampled from $p(x)$, $n = 1, \dots, N$ and generated from $\pi(x)$, the proposal distribution. These samples also have weights associated with them, the importance weights, that allow $p(x)$ to be characterized. Equation 2.14 and 2.15 define the weighted approximation of $p(x)$ and the importance weights, where δ is the Dirac delta function [2, p. 178].

$$p(x) = \sum_{n=1}^N \omega^n \delta(x - x^n) \quad (2.14)$$

$$\omega^n \propto \frac{p(x)}{\pi(x)} \quad (2.15)$$

Applying this to the research, the desired posterior DPD is defined by Equation 2.16, while the importance weights for the samples, $\mathbf{x}_{0:t}^n$, are defined by Equation 2.17[2, p. 178].

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \sum_{n=1}^N \omega_t^{(n)} \delta(\mathbf{x}_{0:t} - \mathbf{x}_{0:t}^{(n)}) \quad (2.16)$$

$$\omega_t^{(n)} \propto \frac{p(\mathbf{x}_{0:t}^{(n)}|\mathbf{y}_{1:t})}{\pi(\mathbf{x}_{0:t}^{(n)}|\mathbf{y}_{1:t})} \quad (2.17)$$

However, a key concern with IS is that it requires all measurements $\mathbf{y}_{1:t}$ before estimating $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$. Thus, for each new measurement that becomes available, \mathbf{y}_{t+1} , the importance weights must be recalculated over the entire state sequence. This aspect renders IS inadequate for recursive estimation. Naturally, the computational requirements and complexity increase with time, either resulting in a lagging filter or inaccurate results [11, p. 9].

2.5.4 Sequential Importance Sampling.

Sequential Importance Sampling (SIS) solves the concerns of IS by modifying IS so that an estimate of $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ may be determined without modifying the previous state sequence $\{\mathbf{x}_{0:t-1}^{(n)}; n = 1, \dots, N\}$. The proposal distribution is factorized according to Equation 2.18[11, p. 9].

$$\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \pi(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})\pi(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t}) \quad (2.18)$$

Iterating Equation 2.18 results in the expression seen in Equation 2.19[11, p.9].

$$\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \pi(\mathbf{x}_0) \prod_{k=1}^t \pi(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k}) \quad (2.19)$$

This importance function allows the weights to now be evaluated recursively in time, as demonstrated in Equation 2.20[11, p. 9] [2, p. 178].

$$\hat{\omega}_t^{(n)} \propto \hat{\omega}_{t-1}^{(n)} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(n)})p(\mathbf{y}_t|\mathbf{x}_{t-1}^{(n)})}{\pi(\mathbf{x}_t^{(n)}|\mathbf{x}_{0:t-1}^{(n)}, \mathbf{y}_{1:t})} \quad (2.20)$$

This process of SIS is simplified further by using the transition prior DPD, $p(\mathbf{x}_{0:t})$, as the proposal distribution, shown in Equation 2.21[11, p.9].

$$\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) \propto p(\mathbf{x}_{0:t}) = p(\mathbf{x}_0) \prod_{i=1}^t p(\mathbf{x}_i|\mathbf{x}_{i-1}) \quad (2.21)$$

By using the transition prior DPD, only \mathbf{x}_t^i must be stored, both the path $\mathbf{x}_{0:t-1}^i$ and history of measurements, $\mathbf{y}_{1:t-1}$ may be discarded. When the filtering distribution, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is the posterior PDF of interest, the proposal distribution only depends on \mathbf{x}_{t-1} and \mathbf{y}_t . The filtering distribution and importance weights may be approximated using Equation 2.22[2, p. 178] and 2.23[11, p. 10].

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{n=1}^N \omega_t^n \delta(\mathbf{x}_t - \mathbf{x}_t^n) \quad (2.22)$$

$$\omega_t^{(n)} \propto w_{t-1}^{(n)} p(\mathbf{y}_t|\mathbf{x}_t^{(n)}) \quad (2.23)$$

One problem encountered by SIS is that as t increases, the distribution of importance weights becomes skewed as preference is given to higher weights. Eventually, one particle has a non-zero importance weight. Thus, the SIS fails to accurately represent the posterior PDFs of interest, namely the filtering distribution. An additional re-sampling step, known as bootstrapping, is used to help mitigate this collapse [11, p.10].

2.5.5 Particle Filter Process.

The particle filter evaluated for this research is known as a Bootstrap Particle Filter (BPF) [2, p. 178]. The process it uses is based on SIS, however it adds an additional step to address the skewed weights present in the SIS. The basic concept of the BPF is to eliminate particles with low importance weights and increase the number of particles with high importance weights [13]. The BPF can be broken into three basic steps, each with several sub-steps. The three steps are: initialization, importance sampling, and selection [11, p. 11]. Figure 2.3 provides a visual depiction of the BPF process in Section 2.5.5.5. The steps within Figure 2.3 are referenced within the three steps.

2.5.5.1 Initialization.

Before the BPF can begin filtering measurements, an initial DPD must be generated. This initial DPD is based on the initial conditions of the state-space model in Equation 2.2. The particles are distributed based on state noise, w or system noise variance, $\sigma^2(\mathbf{x})$. The initialization occurs at timestep $t = 0$.

- For $n = 1, \dots, N_s$, sample $\mathbf{x}_0^{(n)} \sim p(\mathbf{x}_0)$ and set $t = 1$ [11, p. 11]

2.5.5.2 Importance Sampling.

The importance sampling is where the proposal distribution is sampled and is Step 1 on Figure 2.3. As mentioned in Section 2.5.4, the proposal distribution is the prior DPD, which is the probability distribution that expressed uncertainty about the states before measurements are taken into account. It attributes a degree of uncertainty, rather than

randomness, to the uncertain states. This uncertainty is expressed via the system noise variance. $\tilde{\mathbf{x}}$ represent the unweighted predicted states or proposal distribution.

- For $n = 1, \dots, N$, sample $\tilde{\mathbf{x}}_t^{(n)}$ from $p(\mathbf{x}_t | \mathbf{x}_{t-1}^{(n)})$ and set $\tilde{\mathbf{x}}_{0:t}^{(n)} = (\mathbf{x}_{0:t-1}^{(n)}, \tilde{\mathbf{x}}_t^{(n)})$ [11, p. 11]

The importance sampling step also uses the system noise variance to add variety to the proposal distribution, reflected in Step 4 on Figure 2.3. Variety must be added after the re-sampling step since re-sampling creates multiple samples with the same states, in essence, reducing the number of unique states. This aspect is further discussed in Section 2.5.5.3 and 2.5.5.4. Step 1 on Figure 2.3 assumes that variety was introduced previously. The importance weights are then sampled. These importance weights are approximations of the relative posterior probabilities of the particles. In other-words, each weight represents the likelihood that its corresponding particle (containing the estimated states) is correct compared to the true states.

- For $n = 1, \dots, N$, evaluate the importance weights [11, p. 11]

$$\omega_t^{(n)} = p(\mathbf{y}_t | \tilde{\mathbf{x}}_t^{(n)}) \quad (2.24)$$

The importance weights are then normalized, for $n = 1, \dots, N$ [11, p. 11].

$$\hat{\omega}_t^{(n)} = \frac{\omega_t^{(n)}}{\sum_{n=1}^N \omega_t^{(n)}} \quad (2.25)$$

2.5.5.3 Selection.

The BPF resamples the particles based on their weights using a weighted roulette selection [11, p. 10] and is Step 3 on Figure 2.3. Thus, particles with greater weights are apt to be sampled more frequently. Different re-sampling algorithms have been developed, though this research relies on the algorithm developed by Gordon et al., which is one of the most popular, and is described in Section 2.5.5.4 [13] [11, p. 10]. Regardless of the resampling algorithm chosen, all follow a similar method.

- Resample with replacement N particles $(\mathbf{x}_{0:t}^{(n)}; i = 1, \dots, N)$ from the set $(\tilde{\mathbf{x}}_{0:t}^{(n)}; n = 1, \dots, N)$ according to the importance weights [11, p. 11]
- Set $t = t + 1$ and proceed to the importance sampling step

As for the actual estimated states, \mathbf{x}_t , these can be determined by a variety of statistical methods that analyze the posterior DPD, $p(\mathbf{x}_t | \mathbf{y}_{1:t})$. For this research, a mean was taken of the estimated states.

2.5.5.4 Re-sampling Algorithm.

After the importance weights are normalized, a random number, r_u , is sampled from a standard uniform distribution between 0 and 1, which matches the range of normalized importance weights. The normalized weights are summed cumulatively until their cumulative sum is greater than r_u . The index, i , of the last normalized importance weight summed is used to retrieve the predicted state vector corresponding to that weight. This state vector becomes the first discrete point in the posterior DPD. This process is repeated, using a new r_u each time, for the total number of particles, N , at which point the posterior DPD is created. Equation 2.26 describes these steps, where t is the time-step and n is the particle number.

$$\begin{aligned}
\bar{\omega}_t[n] &= \sum_{m=1}^n \hat{w}_m \\
i &= \operatorname{argmax}_{i=1, N} \bar{\omega}[i] : \bar{\omega}[i] > r_u \\
\mathbf{x}_t^{(n)} &= \tilde{\mathbf{x}}_t^{(i)}
\end{aligned} \tag{2.26}$$

As a consequence of the re-sampling step, multiple particle samples are assigned to the same state vector, since the objective of the re-sampling step is to eliminate particles with low importance weights and multiply particles with high importance weights [11, p. 10]. Without introducing variety, the particle filter would begin to suffer from weight degeneration and eventually succumb to weight collapse. Weight degeneration and weight collapse are discussed further in Section 2.5.6. To help prevent weight degeneration, the

sampling step assigns variety to the DPD created by the re-sampling step, seen in Step 4 on Figure 2.3.

2.5.5.5 Visual Depiction of Particle Filter Process.

Figure 2.3 provides a visual illustration of the BPF [11, 12].

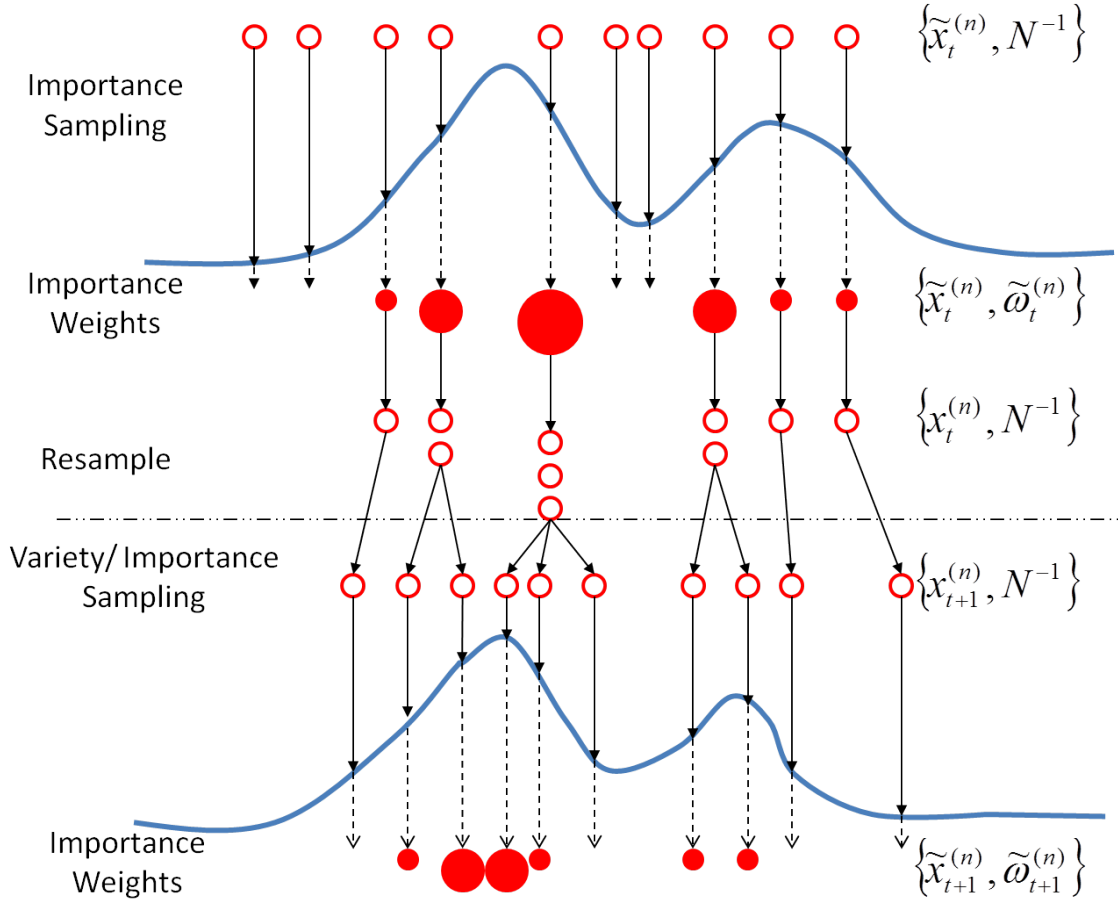


Figure 2.3: Bootstrap Particle Filter Process [11, 12]

In Figure 2.3, $\tilde{x}_t^{(n)}$ represents the population of samples, and N^{-1} , the weights or PMF. The BPF starts at a time-step t with an unweighted measure or importance sampling distribution $\{\tilde{x}_t^{(n)}, N^{-1}\}$ which is an approximation of $p(\mathbf{x}_t | \mathbf{y}_{1:t-1})$. The importance sampling distribution already incorporates variety. Importance weights are calculated using the

measurements at time t , which is an approximation of $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, the posterior PDF of interest from which statistical metrics are computed to generate an estimate of the states. The re-sampling step then selects on the fittest or most heavily weighted particles to create the importance sampling distribution $\{\tilde{\mathbf{x}}_t^{(n)}, N^{-1}\}$. Note, this is still an approximation of $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, the particles are simply reassigned to reflect the current weights. Variety is introduced with the state update at time-step $t + 1$. Customarily, and for this research, the variety is provided using the system noise variances. These variances are tailored to the particle filters used in this research and discussed in Chapter 4 and 5. This variety results in the new importance sampling distribution, $\{\tilde{\mathbf{x}}_t^{(n)}, N^{-1}\}$, which is an approximation of $p(\mathbf{x}_t|\mathbf{y}_{1:t+1})$.

2.5.6 Limitations of the Particle Filter.

Beyond the increased computational needs, particle filters do have several limitations. One of the most common limitations is weight disparity which may lead to weight collapse [11, p.10]. Weight degeneration, occurs when all but a few of the importance weights are close to zero. When most of the importance weights are close to zero, the particle filter cannot produce a good posterior density since only a few wights will be sampled repeatedly subsequently skewing the mean. This degeneration continues to worsen as the mean becomes more inaccurate with each time step until weight collapse occurs once all but one of the importance weights are close to zero. Most computer algorithms crash once the sole particle not close to zero approaches zero likelihood (due to predicted measurements drifting farther from actual measurements) and the algorithm attempts to normalize the weight(s) by dividing by zero. Various methods can mitigate weight collapse, the most common being a re-sampling step whenever the number of effective particles drops below a certain threshold. The number of effective particles can not be directly determined, but may be estimated using Equation 2.27 [2, p.179].

$$\hat{N}_{eff} = \frac{1}{\sum_{n=1}^N (\hat{\omega}_t^{(n)})^2} \quad (2.27)$$

where $\hat{w}_t^{(n)}$ is the normalized weight. \hat{N}_{eff} is less than N and a small N_{eff} indicates severe degradation. More sophisticated particle filters, such as the Sequential Importance Resampling (SIR) filter, Auxiliary Sampling Importance Re-sampling (ASIR) filter, and Regularized Particle Filter (RPF), also seek to eliminate weight collapse through a variety of methods, though for the purposes of this research, these filters were not investigated nor implemented [2, p. 180-183]. Particle filters also depend upon the model of the system in question. Although the particle filter can compensate for noisy measurements and non-linear systems, if the model is sufficiently poor, the filter is unable to perform adequately.

2.5.7 Uses of the Particle Filter.

Due in part to its newness, the particle filter has not been widely implemented and is still in a development stage. Thus, most of the users of the particle filter are not commercial applications, but experimental research. The particle filter and its variants have been used for facial recognition and tracking under varying light and background situations [16]. Additional research at the University of Florence resulted in a first order tracking particle filter for targets moving in two dimensions [6]. Research sponsored by the U.S. Army Research Laboratory and U.K. Ministry of Defense also explored using Bayesian filtering to track multiple targets via proximity sensors [15]. Although much of the research utilizing the particle filter is concerned with target tracking, the particle filter may be applied to a variety of situations with unknown states. One such area is determining the remaining useful life prediction of lithium-ion batteries [17].

2.6 Conclusion

Although the particle filter has seen limited application for tracking, discussed in Section 2.5.7, no research indicated any attempt to use the particle filter to track a target within 3-D space based on the position and movement of the target's centroid using only position measurements. Additionally, no research indicated any attempt to use the particle filter to estimate depth without actual depth measurements.

III. Methodology: Prototype Filters

3.1 Introduction

This chapter details the prototype particle filters developed to test and evaluate the potential to use particle filters to determine the hidden states of a moving target using camera measurements.

3.2 System Design Approach

Although the concept of the particle filter predates more mature filters such as the Kalman filter, development of the particle filter lags behind other filters. Thus, the first stage is to develop a prototype particle filter within the MATLAB environment and determine if hidden states can successfully be determined based on limited measurements. Once verified, particle filters for both a dual camera system and single camera system will be developed. Included in this stage is the development of a target model to both generate simulated measurements for the particle filter and to be used within the particle filter to generate particle distributions. The target model will differ slightly for each camera setup, dual and single respectively. Two prototype particle filters, A and B, were developed to evaluate various aspects of the particle filter.

3.3 Prototype Particle Filter A: Single Variable Tracking

The first filter developed, PPF-A, was based on the filter created by Godon et al. [13]. The primary purpose of PPF-A was to serve as a software validation of the particle filter within the MATLAB environment. PPF-A will also serve as the framework on which all subsequent particle filters will be developed. PPF-A uses the same state, x_t , and measurement, y_t , models as the filter created by Gordon et al. The same noise covariances in the state, w_k , and measurement, v_t were also used. Equation 3.1 and 3.2 describe the

model used.

$$x_t = 0.5x_{t-1} + \frac{25x_{t-1}}{1 + x_{t-1}^2} + 8 \cos(1.2(t - 1)) + w_t \quad (3.1)$$

$$y_t = \frac{x_t^2}{20} + v_t \quad (3.2)$$

The variables w_t and v_t are zero-mean Gaussian white noise with state and measurement variances of 10 and 1 respectively. Additionally, the number of particles, N , used was 500 and the initial state was $x_0 = 0.1$. An example run is seen in Figure 3.1.

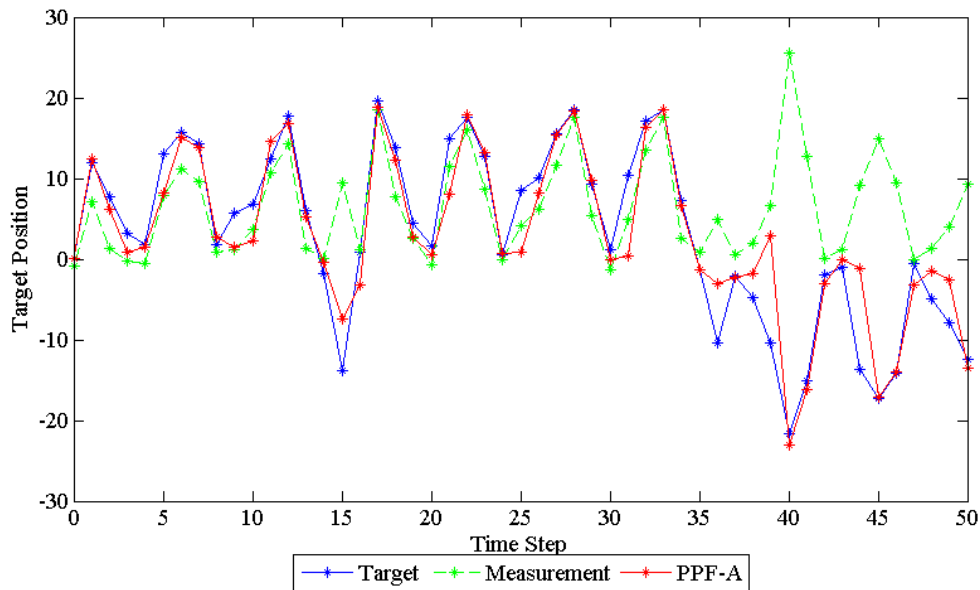


Figure 3.1: Prototype Particle Filter A: Non-linear Function Demonstration

Figure 3.1 demonstrates the particle filter's ability to track a non-linear target function given noisy measurements. The ability to track non-linear target is a key difference between the particle filter and other filters such as the Kalman; the particle filter accepts higher order estimates of perfect order. Perfect order estimates are estimates that are not truncated or linearized as they must be in the Kalman filter. Due to the highly non-linear nature of the chosen target function, seen in Equation 3.1, a Kalman filter was not used as a

comparison, however future functions within Chapter 3, 4, and 5 will use the Kalman filter as a comparison to the particle filter.

3.4 Prototype Particle Filter B: Multi-Variable Tracking and Hidden State Determination

PPF-B evaluated the particle filter's tracking ability with the presence of hidden states; that is, states that can only be predicted within the model since they do not correspond directly to a measurement.

3.4.1 Target Model.

The target model used contained both x and y positions as well as a velocity magnitude V and a velocity heading angle θ . Both positions vary with velocity, velocity changes with a constant acceleration, a , and the heading $\dot{\theta}$ is held constant. Additionally, system noise is introduced into the model for both a and $\dot{\theta}$ via $\sigma^2(x_a)$ and $\sigma^2(x_{\dot{\theta}})$, the system noise variances for acceleration and heading velocity respectively. When system noise is introduced, a and $\dot{\theta}$ are no longer constant. The variance for each state, $\sigma^2(\mathbf{x})$, provides a measure of the spread of that state's density function. Often, and for this research, the standard deviation, $\sigma(x)$ is used in-place of the variance to denote the spread of the density function. The addition of system noise simulates the presence of modeled, higher-order functions, such as changes in acceleration. System noise is not added to the target positions u and v since the target positions are derived from V and θ and ultimately the acceleration values, a and $\dot{\theta}$. r_n is a normally distributed random number that is used to generate the system noise. It

changes each time it appears. Equation 3.3 through 3.8 describe the target model.

$$u_{T,t} = u_{T,t-1} + V_{T,t-1} \cdot \sin(\theta_{T,t-1}) \cdot \Delta t \quad (3.3)$$

$$v_{T,t} = v_{T,t-1} + V_{T,t-1} \cdot \cos(\theta_{T,t-1}) \cdot \Delta t \quad (3.4)$$

$$V_{T,t} = V_{T,t-1} + a_{T,t-1} \cdot \Delta t \quad (3.5)$$

$$\theta_{T,t} = \theta_{T,t-1} + \dot{\theta}_{T,t-1} \cdot \Delta t \quad (3.6)$$

$$a_{T,t} = a_{T,t-1} + \sqrt{\sigma^2(x_{T,a})} \cdot r_n \quad (3.7)$$

$$\dot{\theta}_{T,t} = \dot{\theta}_{T,t-1} + \sqrt{\sigma^2(x_{T,\theta})} \cdot r_n \quad (3.8)$$

3.4.2 Measurements.

The measurements passed to the filter are the positions u and v . Measurement noise is added to simulate noisy measurements the filter would receive in reality. This noise is represented by the measurement noise variances, $\sigma^2(m_{T,u})$ and $\sigma^2(m_{T,v})$, for u_T and v_T respectively. The equations representing these measurements are:

$$u_{m,t} = u_{T,t} + \sqrt{\sigma^2(m_{T,u})} \cdot r_n \quad (3.9)$$

$$v_{m,t} = v_{T,t} + \sqrt{\sigma^2(m_{T,v})} \cdot r_n \quad (3.10)$$

3.4.3 Particle Filter Model.

The PPF-B model used mimics the target model, however it does not contain a specific acceleration variable since acceleration is unknown and the motivation is to determine velocity and heading states. Rather, any changes in acceleration are treated as system noise. Like the target model, the filter contains its own system noise variances, $\sigma^2(x_{f_v})$ and $\sigma^2(x_{f_\theta})$, for V and θ , to simulate those present in the model. Again, these system noise variances represent the target acceleration values. Ideally these will match the actual noise variances of the target model. These variances are also what the filter uses to generate its

distribution of particles.

$$u_{f,t} = u_{f,t-1} + V_{f,t-1} \cdot \sin(\theta_{f,t-1}) \cdot \Delta t \quad (3.11)$$

$$v_{f,t} = v_{f,t-1} + V_{f,t-1} \cdot \cos(\theta_{f,t-1}) \cdot \Delta t \quad (3.12)$$

$$V_{f,t} = V_{f,t-1} + \sqrt{\sigma^2(x_{f,v})} \cdot r_n \quad (3.13)$$

$$\theta_{f,t} = \theta_{f,t-1} + \sqrt{\sigma^2(x_{f,\theta})} \cdot r_n \quad (3.14)$$

The PPF-B measurements, u_{fm} and v_{fm} , are set equal to the target measurements u_m and v_m . These measurements, \mathbf{y}_f are compared to the target measurements, \mathbf{y}_T . Importance weights, ω , are generated from this comparison for each particle, n , as detailed in Equation 3.15. These weights are based on the probability of the particle measurement being correct given the actual measurement while accounting for measurement noise, represented by the variances $\sigma^2(m_{f,u})$ and $\sigma^2(m_{f,v})$. Ideally, these filter measurement noise variances, $\sigma^2(m_f)$, will match the target measurement noise variances, $\sigma^2(m_T)$. The smaller the measurement covariances are, the more the filter trusts the measurements and vice-versa. A normal or Gaussian distribution was used to generate the weights, however any type of distribution may be used to model the target noise, so long as the same distribution is used within the filter. Both measurements, u_m and v_m were weighted equally, as defined by the weight matrix, Ω_f in Equation 3.16 [13]. By setting the weights equal, this means the filter assumes both the measurements, corresponding to each weight, are of equal importance. Weights are typically set at lower values if their corresponding measurements were less accurate and the user does not want these measurements to adversely affect the state estimation. Weights are typically set at higher values if their corresponding measurements are of greater interest and the user desires the filter pay more attention to these measurements and their corresponding states at the exclusion of others.

$$\omega_t^{(n)} = \frac{1}{\sqrt{2\pi\sigma(\mathbf{m}_f)}} \cdot \text{EXP} \left\{ \frac{-((\mathbf{y}_{f,t} - \mathbf{y}_{T,t}^{(L)}) \cdot \Omega_f \cdot (\mathbf{y}_{f,t} - \mathbf{y}_{T,t}^{(n)})^\top)}{2\sigma(\mathbf{m}_f)} \right\} \quad (3.15)$$

$$\Omega_f = \begin{bmatrix} \Omega(u) & 0 \\ 0 & \Omega(v) \end{bmatrix} \quad (3.16)$$

3.4.4 Kalman Filter Model.

Two Kalman filters serve as comparisons. Prototype Kalman Filter A (PKF-A) uses the same initial conditions as the target and the same acceleration values as the target, which are reflected in the control vector u . Prototype Kalman Filter B (PKF-B) uses the same initial conditions as the PPF-B and slightly different acceleration values, to illustrate the lag that occurs when the control vector is incorrect. Since the Kalman filter must use linear functions, the non-linear functions of velocity and heading are derived from the component velocities of u and v , \dot{u} and \dot{v} respectively, as seen in Equation 3.17 and 3.18.

$$\dot{u}_t = V \cdot \cos(\theta_t) \quad (3.17)$$

$$\dot{v}_t = V \cdot \sin(\theta_t) \quad (3.18)$$

Equation 3.17 and 3.18 are also applied to the control law and covariance matrices when applicable. In order to provide a comparison to the target and PPF-B, \dot{u} and \dot{v} values are transformed to their respective V and θ values using Equation 3.19 and 3.20.

$$V_t = \sqrt{\dot{u}_t^2 + \dot{v}_t^2} \quad (3.19)$$

$$\theta_t = \arctan\left(\frac{\dot{u}_t}{\dot{v}_t}\right) \quad (3.20)$$

Equation 3.21 through 3.26 describe the the state transition model matrix, \mathbf{A} , the control-input model, \mathbf{B} , the control vector, \mathbf{u}_k , the system noise covariance matrix, \mathbf{Q} , the measurement model, \mathbf{C} , and the measurement noise covariance matrix, \mathbf{R} .

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.21)$$

$$\mathbf{B} = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \quad (3.22)$$

$$\mathbf{u}_k = \begin{bmatrix} \ddot{u} \\ \ddot{v} \end{bmatrix} \quad (3.23)$$

$$\mathbf{Q} = \begin{bmatrix} \sigma(x_u) & 0 & 0 & 0 \\ 0 & \sigma(x_v) & 0 & 0 \\ 0 & 0 & \sigma(x_V) & 0 \\ 0 & 0 & 0 & \sigma(x_\theta) \end{bmatrix} \quad (3.24)$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (3.25) \quad \mathbf{R} = \begin{bmatrix} \sigma(y_V) & 0 \\ 0 & \sigma(y_\theta) \end{bmatrix} \quad (3.26)$$

3.4.5 Simulation Scenarios.

Three simulations were conducted, changing various parameters of the model and filter to validate the claim that particle filter could determine hidden states. Initial conditions for the target, PKF-A, PKF-B, and PPF-B are provided in Table 3.1.

Table 3.1: PPF-B Scenario: Initial Conditions

States	u	v	V	θ	a	$\dot{\theta}$
Target	10	10	1	30	1	2
PKF-A	10	10	1	30	1	2
PKF-B	0	0	0	0	0	0
PPF-B	0	0	0	0	N/A	N/A

Additional parameters used throughout the three scenarios are provided in Table 3.2.

Table 3.2: PPF-B Scenario: Parameters

Filter	PKF-A	PKF-B	PPF-B
Number of Simulations	100	100	100
Run Time	100	100	100
Time Step	1	1	1
Number of Particles	N/A	N/A	500

3.4.5.1 Scenario 1: No System or Measurement Noise.

The first scenario contained no system nor measurement noises in the target model. Thus, the measurements observed were completely accurate. Target system variances, $\sigma^2(x_{T,v})$ and $\sigma^2(x_{T,\theta})$ were both 0 as measurement noise variances, $\sigma^2(m_{f,u})$ and $\sigma^2(m_{f,v})$ respectively. Kalman filter system covariance values were set to 0.1 and not 0 for both V and θ in order to propagate the error covariance prediction and the Kalman gain calculation. The measurement covariance values were set to 0 for both measurements u and v . Particle filter system noise covariances were not 0 to match the target, since the filter still must create a distribution of particles, but they were kept low to reflect the lack of system noise with $\sigma^2(x_{f,v})$ and $\sigma^2(x_{f,\theta})$ both equal to 1. The measurement noise variance used by the filter, $\sigma^2(m_f)$, was equal to 0.1 reflecting both the lack of target measurement noise, but also the uncertainty introduced into the filter from the filter system noise covariances. The mean of the difference between each filter value and the target value for all the simulation runs at each time step are calculated and the shown in Figure 3.2. Rather than separate u and v position variables, the Euclidean distance is used instead on the basis that both positions should be accurate. Additionally, the results from a single simulation are provided in Figure 3.3.

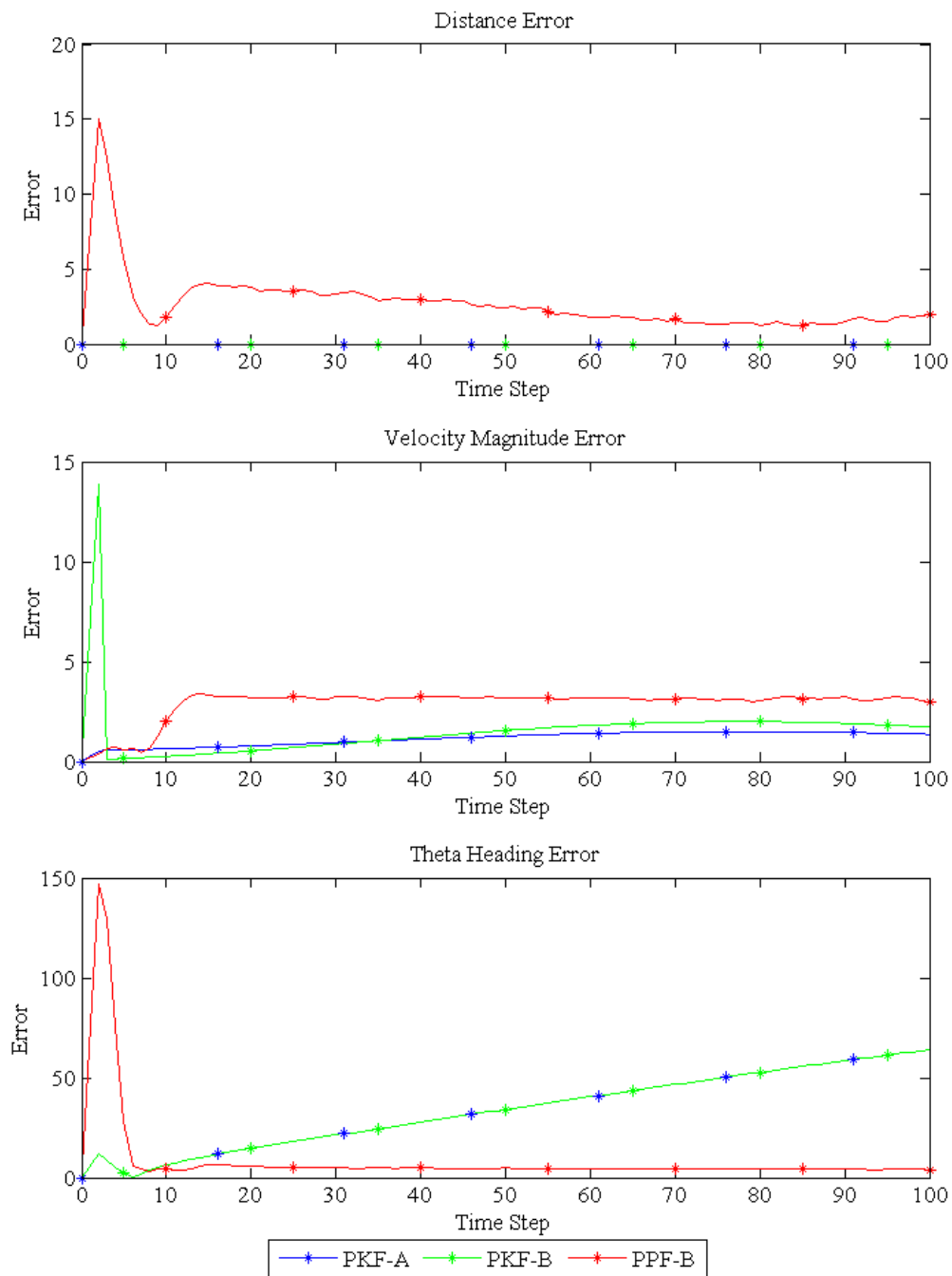


Figure 3.2: PPF-B: Mean Error for No System or Measurement Noise

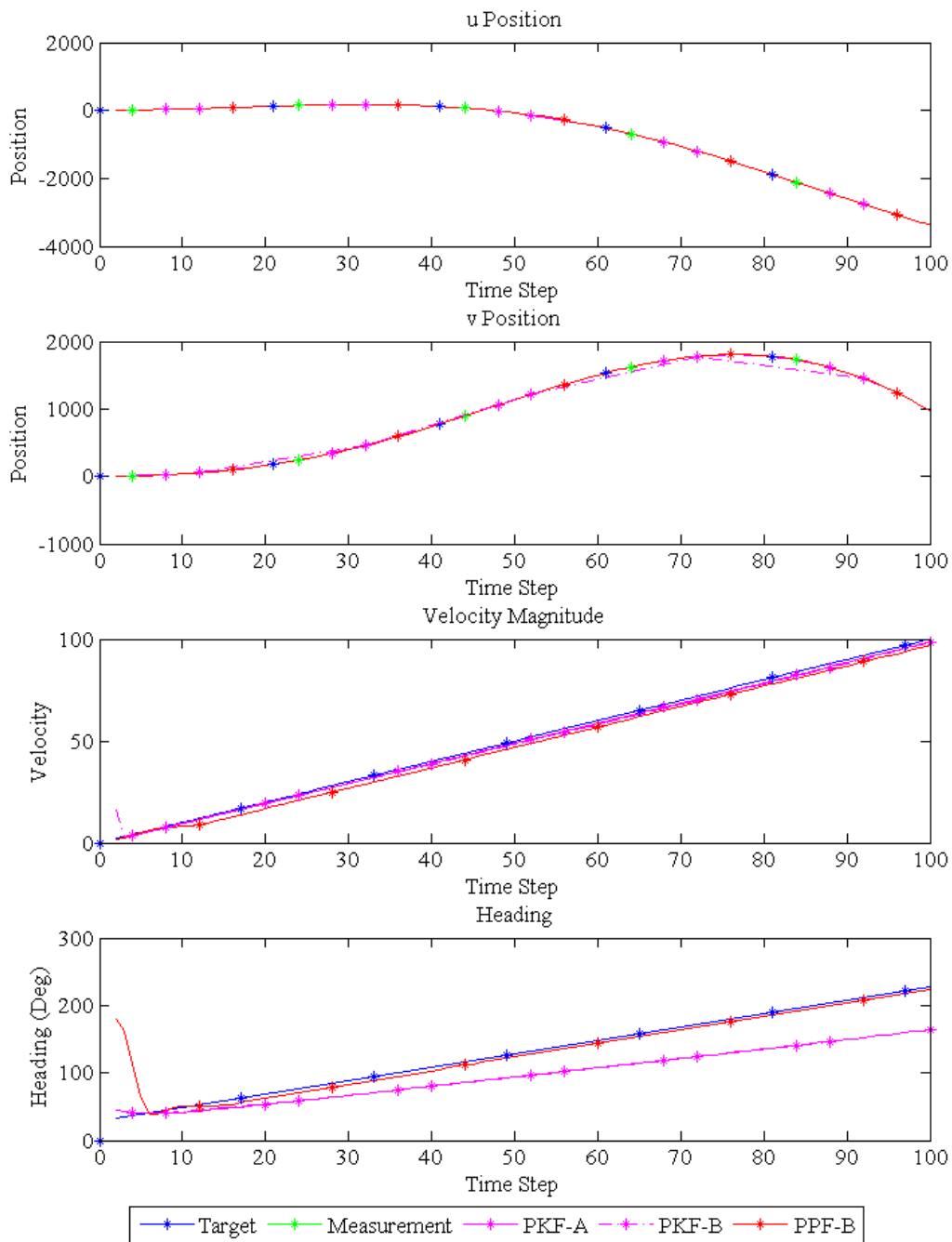


Figure 3.3: PPF-B: Single Run for No System or Measurement Noise

As seen in Figure 3.2 and 3.3, the filter is able to determine the velocity and heading once the filter locks onto the u and v position. Initially there is an overshoot and undershoot of the velocity and heading to reflect the increase in velocity needed to match the simulated measurements to actual measurements since the filter lags behind the target due to differing initial conditions. Scenario 1 does not evaluate the performance of the filter, but functions as a diagnostic on the filter's functionality. As discussed, the filter system and measurement noise covariances cannot be set to zero, nor would they ever since a system with zero noise does not need a filter whose purpose is to remove noise. Unsurprisingly, given the lack of noise and relative linearity of the system, both Kalman filters performed better than PPF-B for position and velocity estimation. However, neither Kalman filter returned accurate heading estimates. This is likely due to the errors in the velocity estimation that are compounded when predicted in heading, as seen in Equation 3.20. Also as expected, velocity errors for PKF-B were greater than PKF-A due to the erroneous control law. Table 3.3 provides the mean error values for each filter for each variable over the last 50 time steps. The last 50 were chosen in order to allow PKF-B and PPF-B to track to the target since neither began with the target's initial conditions.

Table 3.3: Mean Errors for Last 50 Time Steps without System or Measurement Noise

Metrics	u	v	V	θ
PKF-A	0	0	1.4337	49.4796
PKF-B	0	0	1.8665	49.796
PPF-B	1.6738	3.1969	3.1448	4.3499

3.4.5.2 Scenario 2: Measurement Noise, No System Noise.

. Scenario 2 contained measurement noise but no system noise. The presence of measurement noise increases the uncertainty imparted on the filter from the measurements. The variances for system noise are provided in Table 3.4.

Table 3.4: PPF-B Scenario 2: Variances

Variance	$\sigma(y_u)^2$	$\sigma(y_v)^2$
PKF-A	10	10
PKF-B	10	10
PPF-B	10	10

As with Scenario 1, the mean errors were calculated and can be seen in Figure 3.4. The results from a single simulation run are provided in Figure 3.5.

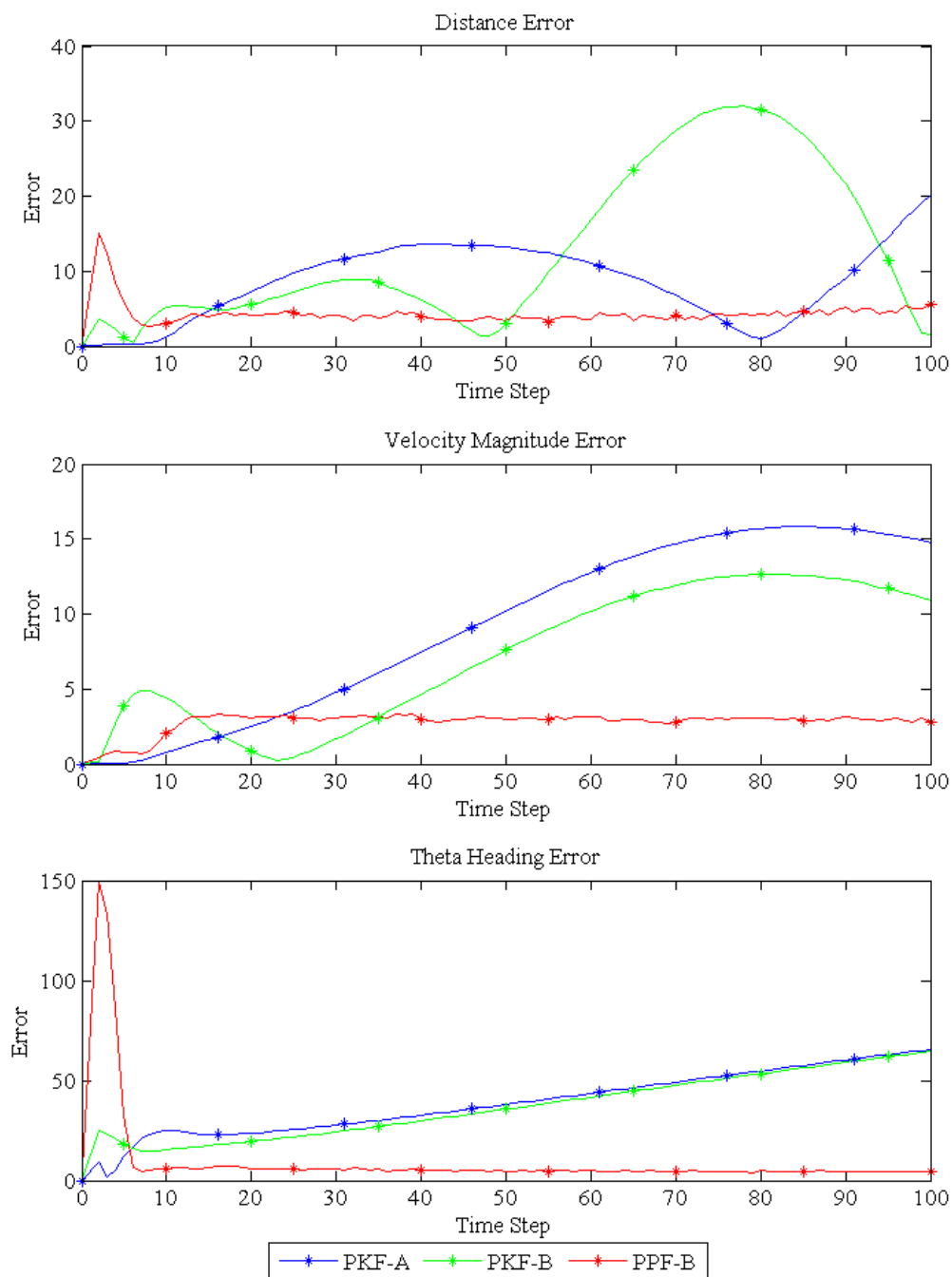


Figure 3.4: PPF-B: Mean Error for No System Noise

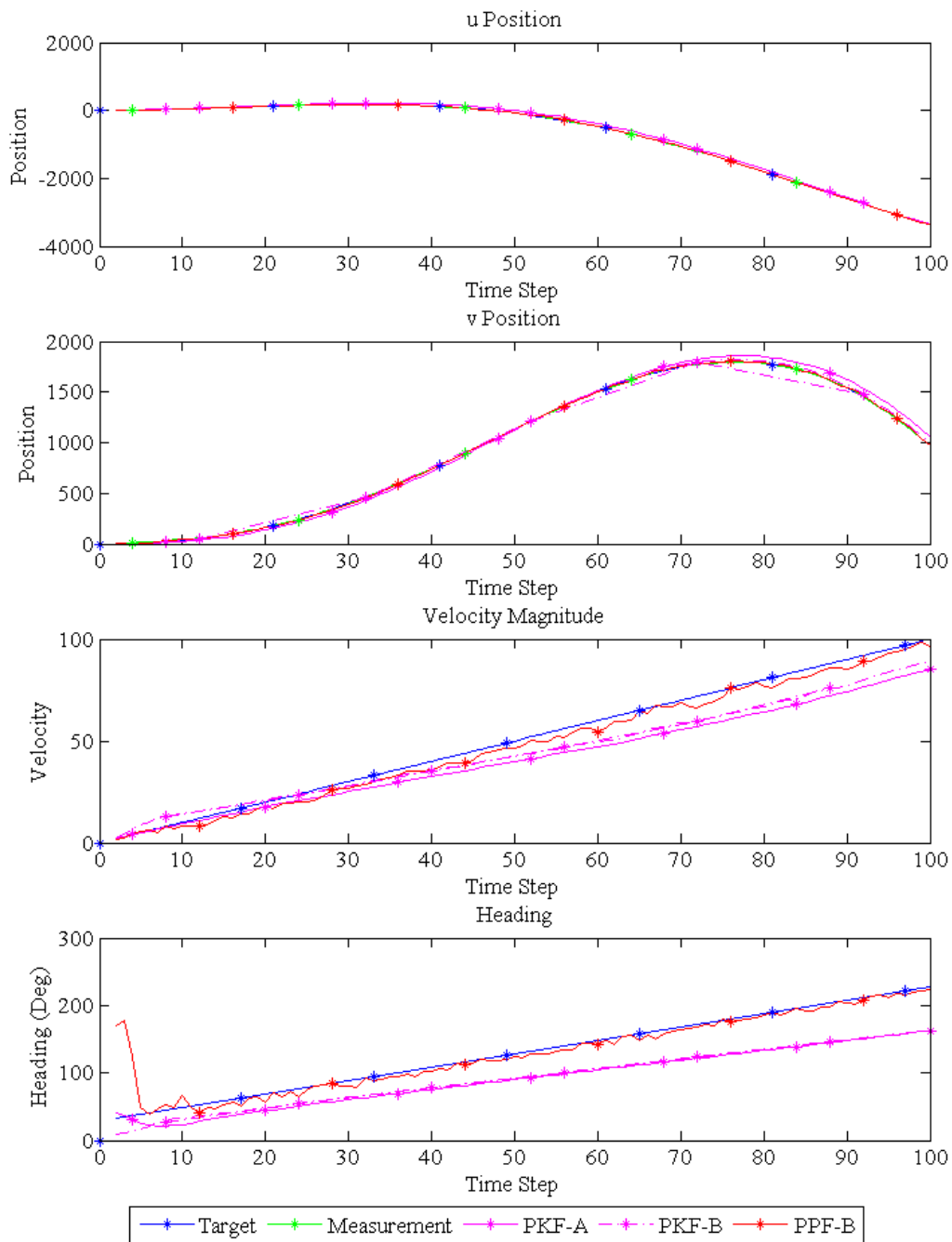


Figure 3.5: PPF-B: Single Run for No System Noise

As seen in Figure 3.5, despite the presence of the system noise, the filter is still able to track the u and v positions of the target as well as provide reasonable estimates of the velocity and heading despite the noticeable non-linearity of the target variables. However, performance for both PKF-A and PKF-B worsens compared to Scenario 1. Although the position estimation is still reasonable for both, estimates of velocity and heading worsen due to their non-linearity. Furthermore, the errors within velocity and heading adversely affect the position estimation. As with Scenario 1, the position estimation performance of PKF-B is worse than the performance of PKF-A due to PKF-B's erroneous control law. The mean errors for the last 50 time steps are provided in Table 3.5.

Table 3.5: Mean Errors without System or Measurement Noise

Metrics	u	v	V	θ
PKF-A	9.0307	49.6103	14.2988	51.8838
PKF-B	20.2419	16.2025	11.2875	50.3598
PPF-B	4.2103	5.3242	2.9838	4.5664

3.4.5.3 Scenario 3: System Noise and Measurement Noise.

. Scenario 3 contained system noise and measurement noise in the target model. While the filter seeks to track changes in the target due to system noise, the filter should not track changes due to measurement noise, but instead filter measurement noise to determine and track the actual states. Both system and measurement noises were identical in the target and filter models. Table 3.6 contains the system and measurement variances used for Scenario 3.

Table 3.6: PPF-B Scenario 3: Variances

Variance	$\sigma(y_u)^2$	$\sigma(y_v)^2$	$\sigma(x_v)^2$	$\sigma(x_\theta)^2$
PKF-A	10	10	1	5
PKF-B	10	10	1	5
PPF-B	10	10	2	10

System variance values, $\sigma(x_v)^2$ and $\sigma(x_\theta)^2$, for PKF-B were greater in order to introduce additional variety due to the lack of an acceleration parameter for PKF-B. The mean errors were calculated and can be seen in Figure 3.6. The results from a single simulation run are provided in Figure 3.7. An important note is that since the system noise directly affects the subsequent target state calculation, each simulation will be different.

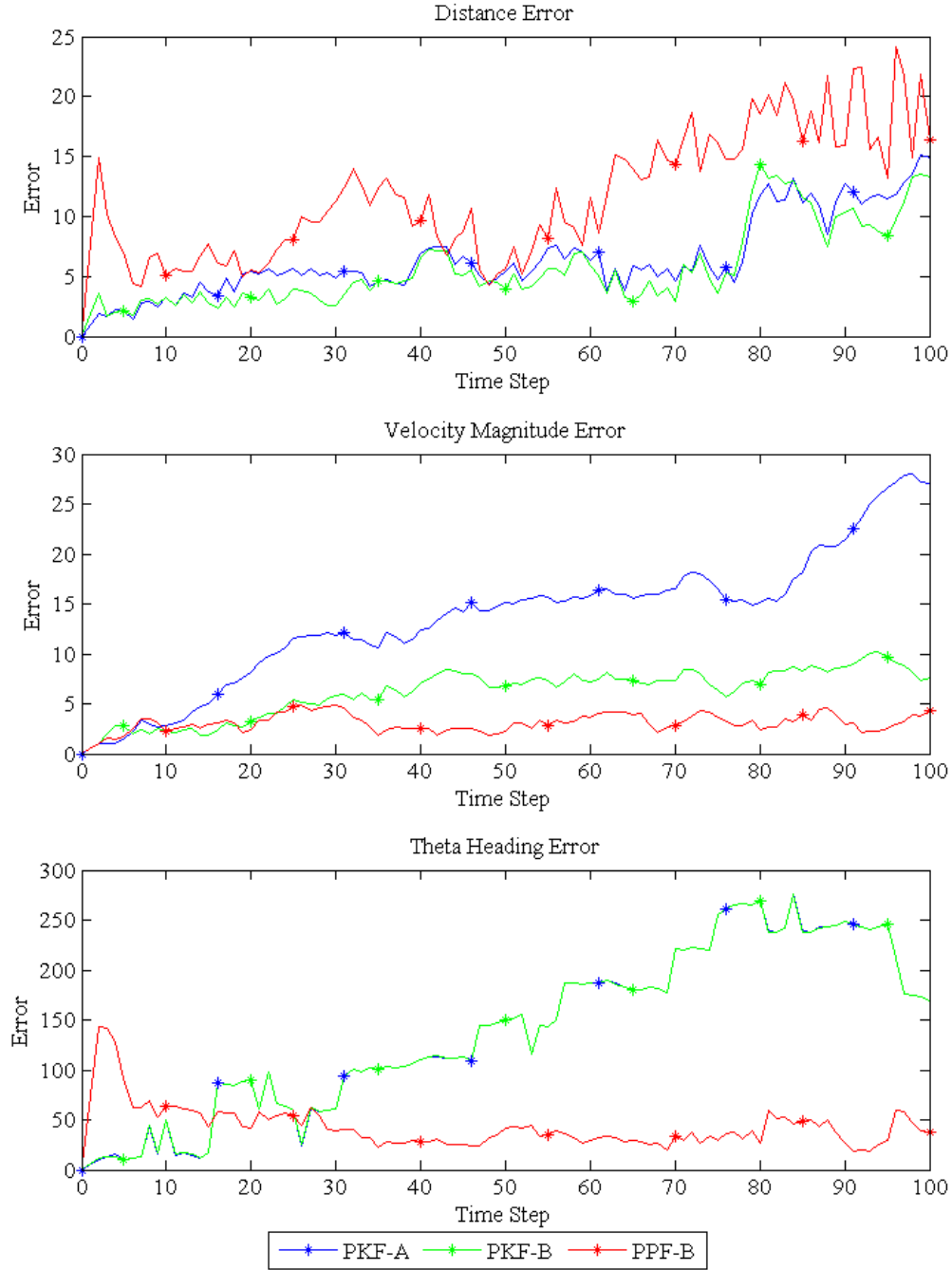


Figure 3.6: PPF-B: Mean Error for System and Measurement Noise

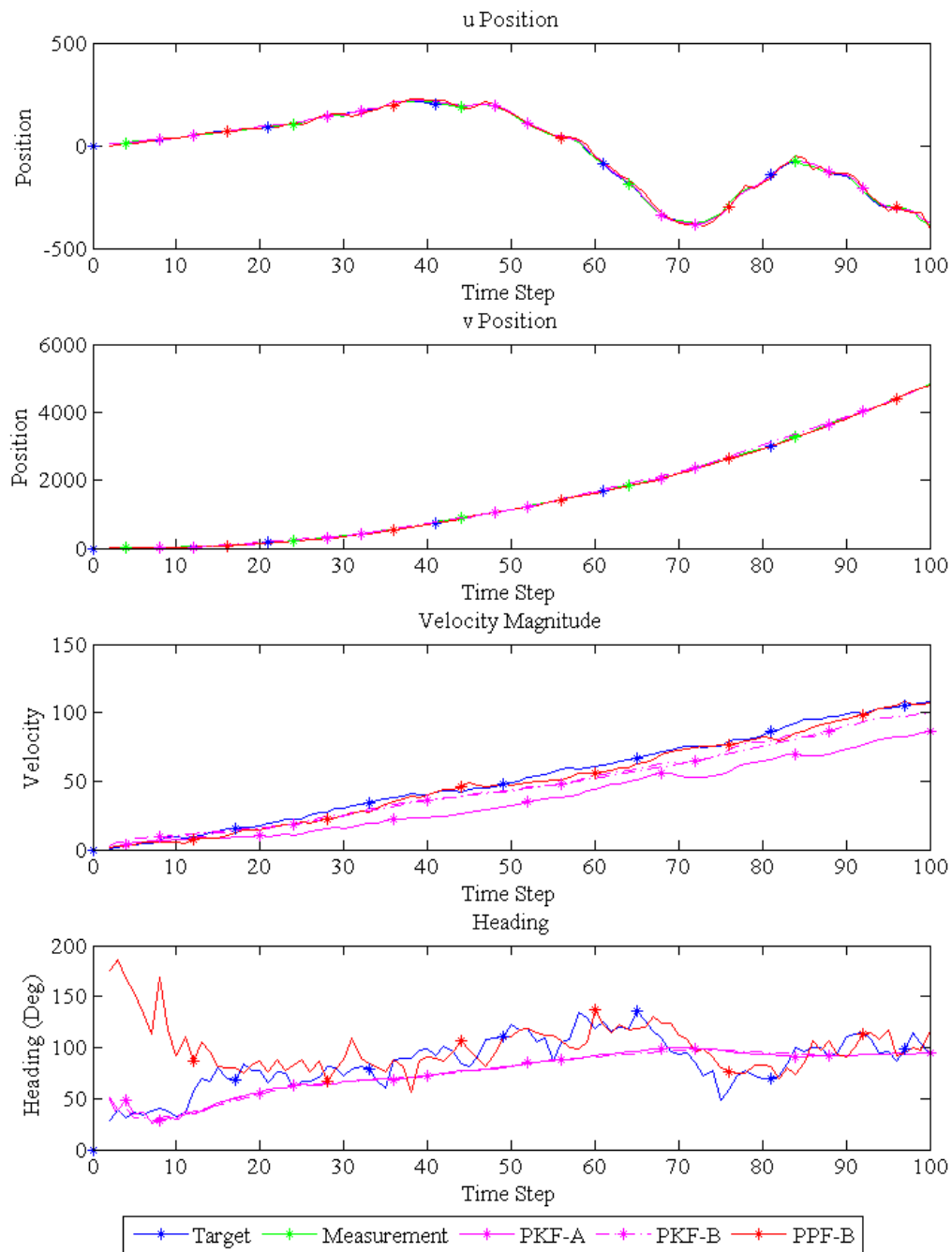


Figure 3.7: PPF-B: Single Run for System and Measurement Noise

As seen in Figure 3.7, with the presence of measurement noise, all filter estimations become less accurate. In particular, over time, PPF-B position estimates are worse than those of PKF-A or PKF-B however PPF-B still returns notably better estimates of velocity and heading. Furthermore, since acceleration values change, neither control law is accurate and is particularly evident again for velocity and heading estimates. The differences for the hidden states, velocity and heading, illustrate a key benefit of the particle filter; the particle filter does not require knowledge or assumptions concerning acceleration. The overall mean values for each filter are provided in Table 3.7.

Table 3.7: Mean Errors without System or Measurement Noise

Metrics	u	v	V	θ
PKF-A	8.4716	14.7124	18.4685	209.1750
PKF-B	7.6093	11.7084	7.8265	209.0449
PPF-B	15.0517	13.8954	3.3029	36.0236

3.4.5.4 Scenario 4: Weight Discrepancy and Collapse.

. Scenario 4 illustrates a weight collapse. Table 3.8 contains the measurement covariances used to produce the weight collapse.

Table 3.8: PPF-B Scenario 2: Variances

Variance	$\sigma(y_u)^2$	$\sigma(y_v)^2$
Target	1000	1000
PPF-B	10	10

By choosing measurement covariances for the target that are greater than those for the filter ensures that some measurement points will lie outside the distribution used by the

filter. The filter will not generate the correct probability that the state is correct given the measurement. The filter assumes the measurements are more accurate, and hence representative of the target states, than they actually are. Although the filter will still use the most heavily weighted and thus correct particle, the total importance weight summation begins to approach zero. Eventually, none of the particles have significant weights resulting in filter collapse as seen in Figure 3.8.

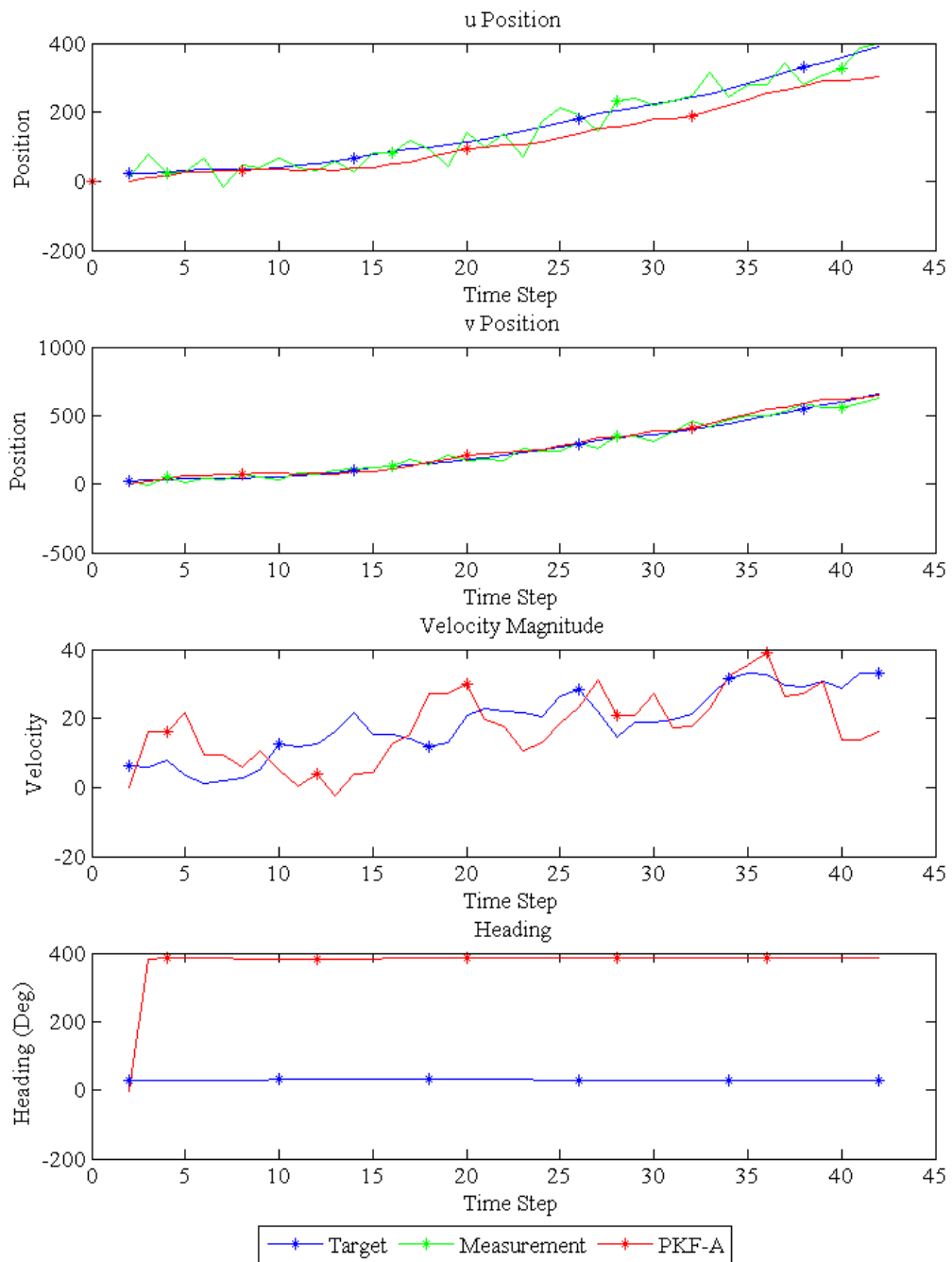


Figure 3.8: PPF-B: Weight Discrepancy and Collapse

Towards the end of the 'u Position' subplot, PPF-B begins to diverge from the target. This illustrates PPF-B's inability to generate and sample particles from a distribution that contains the target solution. Figure 3.9 shows the distribution of particles versus the corresponding target state.

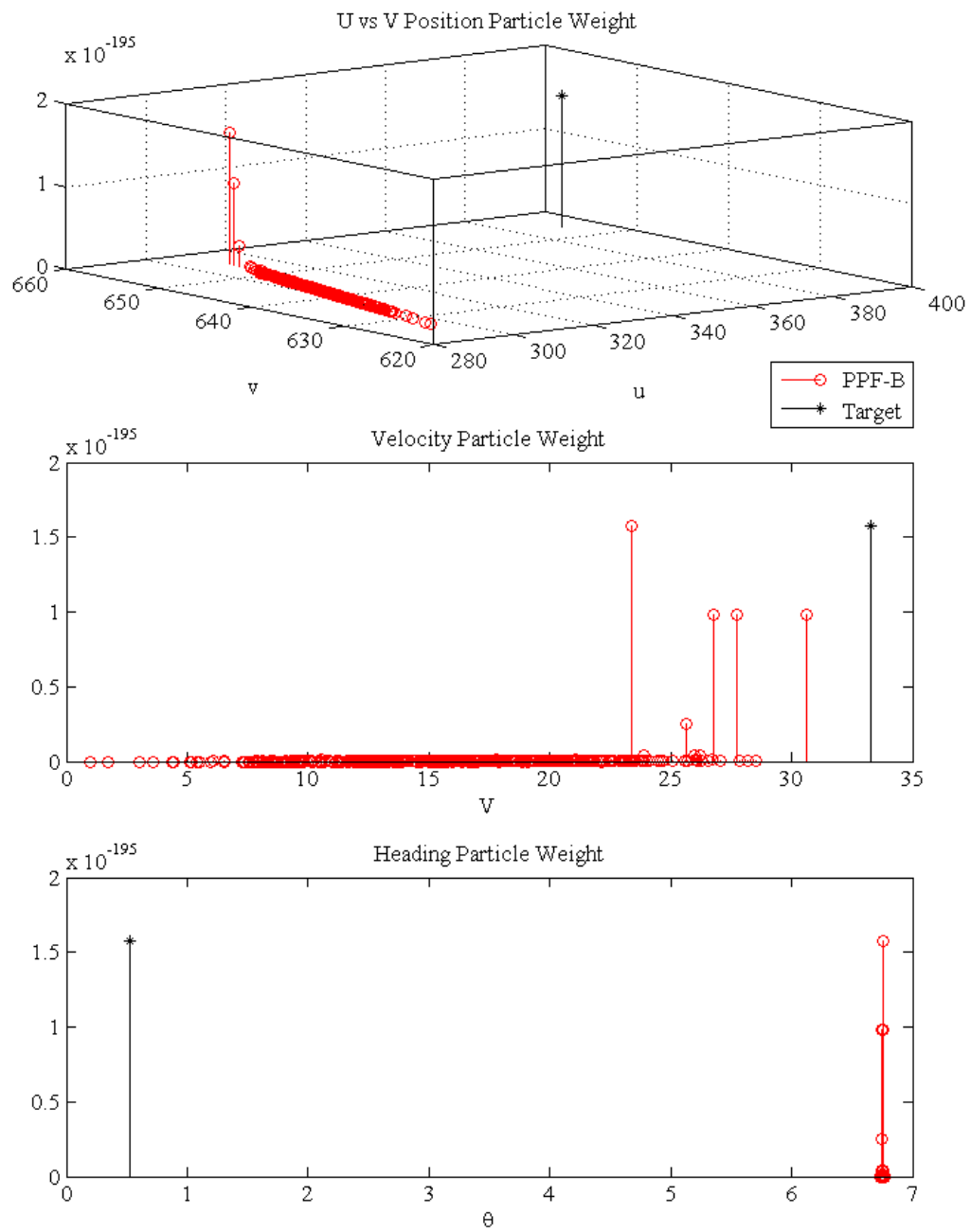


Figure 3.9: PPF-B: Iteration Prior to Weight Collapse

As seen in the first subplot, U vs V Position Particle Weight, the target state lies well outside of the particle distribution. Naturally, PPF-B assigns the greatest weight the closest particle, but the overall weight values are exceptionally small. The collapse occurs on the subsequent iteration when the weights are indistinguishable from zero, meaning PPF-B cannot select any particle. These collapses can be mitigated by increasing the filter's measurement noise covariance to equal or exceed the target measurement noise covariance. If the filter's measurement noise covariance exceeds that of the target, the filter can still track the target assuming a sufficient number of particles. An increase in measurement noise covariance results in an increase in particle distribution. A larger particle distribution results in an increase in the number of particles with an importance weight than would be justified by the actual target measurement noise. In essence, the filter keeps particles that it should discard due to the inaccurate importance weights. Thus, in-order to decrease computational costs, noise covariances should ideally be equal for both the target and filter.

IV. Methodology: Application Filters

4.1 Introduction

This chapter details the filter development philosophy as well as the methods used to evaluate the performance of the particle filter.

4.2 System Design Approach

Proceeding forward based on positive results from the prototype filters discussed in Chapter 2, the subsequent particle filters, EPF-A and EPF-B, attempt to track a target moving non-linearly in 3-D space. Before developing the filters, the motion of the target must be modeled in the three frames discussed in Section 4.2.1: the global frame, the camera frame, and the pixel frame. Additionally, a measurement model will supply simulated camera measurements to the filters using the target model as a basis. EPF-A will be applied to the camera frame and EPF-B will be applied to the pixel frame.

4.2.1 Coordinate Frames.

In order to relate measurement data from the cameras to the target's position in 3-D space, the data must be moved through a series of coordinate frame transformations. The steps necessary to move from the global frame, which the target moves in, to the final pixel frame that the camera uses to observe the target are determined with a series of coordinate frame transformations. These transformations are necessary to both generate simulated measurements and produce the particles by the particle filter. The relationship between the target data and single camera is simpler than the relationship for two cameras. Both the global and camera frames share the same origin simplifying the necessary transformations. The three coordinate frames used are the global frame, the camera frame, and the pixel frame. The first transformation is exclusive to the MATLAB environment due to a difference between the default MATLAB coordinate frame and the preferred global frame

used by imaging systems. To differentiate between the default MATLAB global frame and imaging global frame, the axis labels change using the following notation.

$$X_1 \rightarrow X_G$$

$$Y_1 \rightarrow Y_G$$

$$Z_1 \rightarrow Z_G$$

The default positive axis directions are X_1 up, Y_1 right, and Z_1 into the page. The customary positive axis directions for image processing are X_G right, Y_G down, and Z_G into the page. Thus, a 90° rotation about the Z axis is performed, see Figure 4.1.

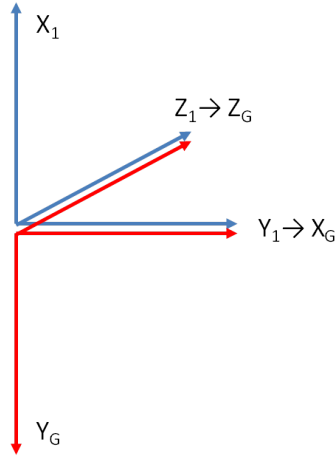


Figure 4.1: Global Transformation from MATLAB Default Orientation

The second transformation is from the global to the camera frame, that is, the frame aligned with the camera. The axis labels change using the following notation.

$$X_G \rightarrow U_1 \rightarrow U_C$$

$$Y_G \rightarrow V_1 \rightarrow V_C$$

$$Z_G \rightarrow W_1 \rightarrow W_C$$

This is how one would perceive the global frame if rotating and panning with the camera. The two angles necessary for the transformation are a pan, α , about the Y_G axis, and a tilt, β , about the W_1 axis, see Figure 4.2.

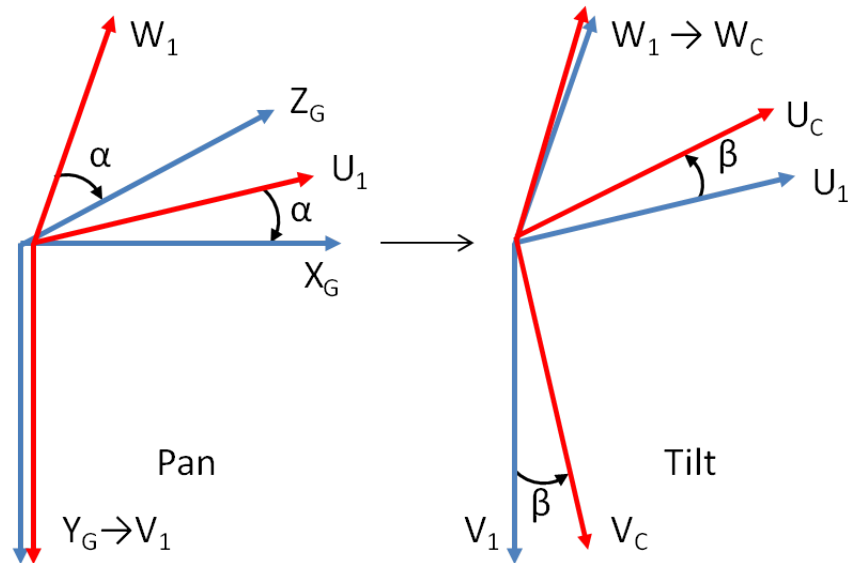


Figure 4.2: Transformation from Global to Camera Frame

These two transformations, from the global to the camera frame, can be described using a Direction Cosine Matrix (DCM), see Equation 4.1. All the rotations detailed thus far are about one of the three orthogonal dimensions, X_1 , Y_1 , and Z_1 .

$$\begin{aligned} \begin{bmatrix} \cos(\alpha - 90) & 0 & -\sin(\alpha - 90) \\ 0 & 1 & 0 \\ \sin(\alpha - 90) & 0 & \cos(\alpha - 90) \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta - 90) & -\sin(\beta - 90) \\ 0 & \sin(\beta - 90) & \cos(\beta - 90) \end{bmatrix} \begin{bmatrix} \cos(90) & -\sin(90) & 0 \\ \sin(90) & \cos(90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \text{Y Rotation} & \text{X Rotation} & \text{Z Rotation} \end{aligned} \quad (4.1)$$

The final transformation is from the camera frame to the pixel frame, that is, the three dimensional target location is projected onto two dimensions. This is accomplished by dividing each U_C and V_C coordinate by the corresponding W_C coordinate, see Equation 4.2.

$$[u_P, v_P] = \frac{1}{w_2} [u_C, v_C] \quad (4.2)$$

Figure 4.3 depicts the transformation from the camera to pixel frame.

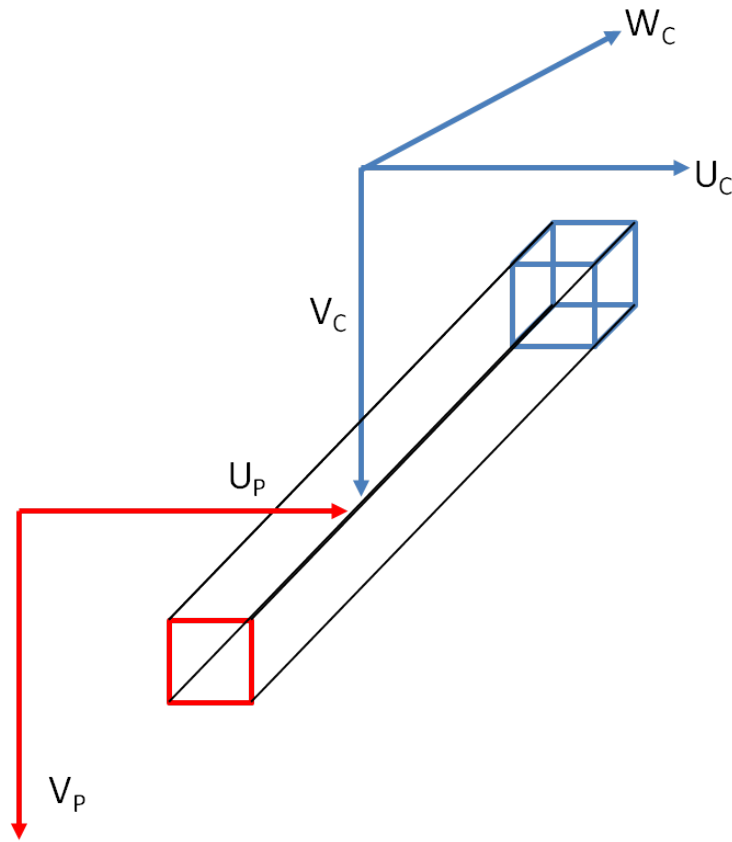


Figure 4.3: Transformation from Camera to Pixel Frame

As seen in Figure 4.3, if an object is viewed without knowing any absolute measurements, it is impossible to determine the size or depth of that object once it is

projected onto a two-dimensional image plane. This is the point at which depth information is lost for a single camera, since w_C cannot be redetermined from u_P and v_P alone.

4.2.2 Target Model.

Based upon reference frames in Section 4.2.1, a second order model of the target was developed. The equations governing the motion of the target model are detailed in Section 4.2.4. The target model generates the movements of a single point representing the centroid of the target. The target is assumed to move within a three dimensional space, with a position of x_T , y_T , and z_T , and a velocity magnitude of V_T defined in the three dimensional space with the angles θ_T , tilt from the Y axis, and ϕ_T , pan in the X-Z plane, as shown in Figure 4.4.

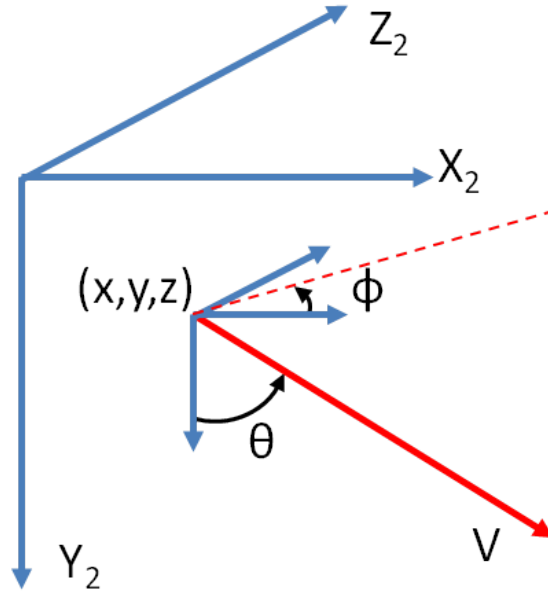


Figure 4.4: Target in Global Frame

The states that describe the position and motion of the target in the global frame are provided in Table 4.1.

Table 4.1: Global Frame Target States

Parameter	Target States		
Position	x_T	y_T	z_T
Motion	V_T	θ_T	ϕ_T
Position Changes	Δx_T	Δy_T	Δz_T
Motion Changes	ΔV_T	$\Delta \theta_T$	$\Delta \phi_T$
Rates of Position	\dot{x}_T	\dot{y}_T	\dot{z}_T
Rates of Motion	\dot{V}_T	$\dot{\theta}_T$	$\dot{\phi}_T$

4.2.3 Measurement Model.

Using the point generated by the target model as a basis, the measurement model generates additional points around the target model that move in relation to the target model, simulating a three-dimensional object moving in three-dimensional space. The purpose of generating additional points is to simulate the noise that may be introduced when the model attempts to determine the motion of the centroid based on these additional points. Eight measurement points were generated, forming a cube around the target centroid as shown in Figure 4.5.

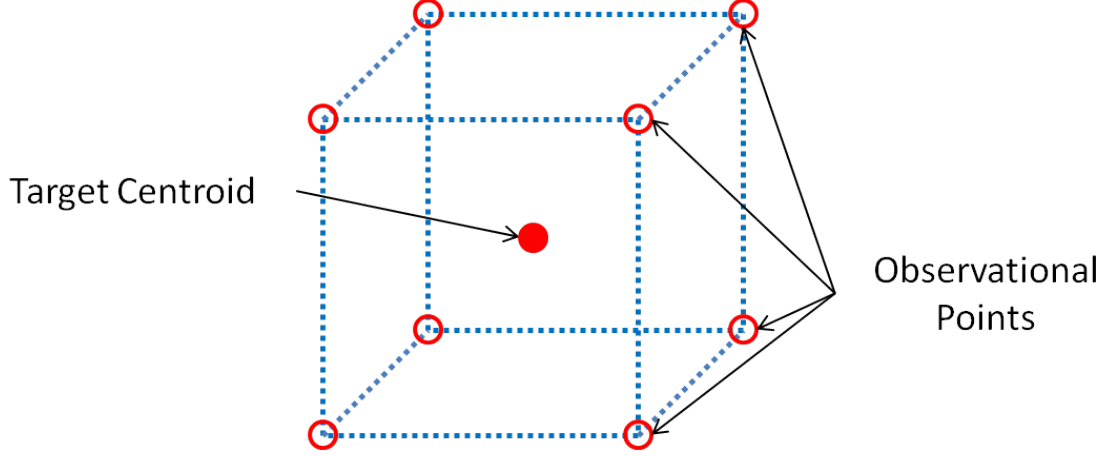


Figure 4.5: Target Centroid and Measurement Points

4.2.4 Global Frame Equations.

The motion of the target in the global frame, and indeed all frames, is governed by the changes in V and headings θ and ϕ . These variables define the position of the target in the global frame using the states x , y , and z . Additionally, the motion variables are impacted by their respective rates of change, \dot{V}_T , $\dot{\theta}_T$, and $\dot{\phi}_T$. The target model is a second-order discrete time system, derived by evaluating the continuous system at a time Δt and eliminating all terms that greater than second-order. The knowledge of acceleration is what allows the possibility of determining depth, discussed further in Section 4.2.7 and 4.4. The position of the target is defined by Equation 4.3 through 4.5.

$$x_{T,t} = x_{T,t-1} + \Delta x_{T,t-1} \quad (4.3)$$

$$y_{T,t} = y_{T,t-1} + \Delta y_{T,t-1} \quad (4.4)$$

$$z_{T,t} = z_{T,t-1} + \Delta z_{T,t-1} \quad (4.5)$$

Δx_T , Δy_T , and Δz_T reflect the change in position with each time step. The changes in position are determined by the changes in V and headings θ and ϕ and defined by Equation 4.6 through 4.8.

$$\begin{aligned}
\Delta x_{T,t} = & V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \cos(\phi_{T,t}) \cdot \Delta t \\
& + \dot{V}_{T,t} \cdot \sin(\theta_{T,t}) \cdot \cos(\phi_{T,t}) \cdot \frac{\Delta t^2}{2} + \dot{\theta}_{T,t} \cdot V_{T,t} \cdot \cos(\theta_{T,t}) \cdot \cos(\phi_{T,t}) \cdot \frac{\Delta t^2}{2} \\
& - \dot{\phi}_{T,t} \cdot V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \sin(\phi_{T,t}) \cdot \frac{\Delta t^2}{2}
\end{aligned} \tag{4.6}$$

$$\Delta y_{T,t} = V_{T,t} \cdot \cos(\theta_{T,t}) \cdot \Delta t + \dot{V}_{T,t} \cdot \cos(\theta_{T,t}) \cdot \frac{\Delta t^2}{2} - \dot{\theta}_{T,t} \cdot V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \frac{\Delta t^2}{2} \tag{4.7}$$

$$\begin{aligned}
\Delta z_{T,t} = & V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \sin(\phi_{T,t}) \cdot \Delta t \\
& + \dot{V}_{T,t} \cdot \sin(\theta_{T,t}) \cdot \sin(\phi_{T,t}) \cdot \frac{\Delta t^2}{2} + \dot{\phi}_{T,t} \cdot V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \cos(\phi_{T,t}) \cdot \frac{\Delta t^2}{2} \\
& + \dot{\theta}_{T,t} \cdot V_{T,t} \cdot \sin(\theta_{T,t}) \cdot \sin(\phi_{T,t}) \cdot \frac{\Delta t^2}{2}
\end{aligned} \tag{4.8}$$

The velocity and headings at each time step are defined by Equation 4.9 through 4.11.

$$V_{T,t} = V_{T,t-1} + \Delta V_{T,t-1} \tag{4.9}$$

$$\theta_{T,t} = \theta_{T,t-1} + \Delta \theta_{T,t-1} \tag{4.10}$$

$$\phi_{T,t} = \phi_{T,t-1} + \Delta \phi_{T,t-1} \tag{4.11}$$

The changes in velocity and heading from each time step are defined by Equation 4.12 through 4.14.

$$\Delta V_{T,t} = \dot{V}_{T,t} \cdot \Delta t \tag{4.12}$$

$$\Delta \theta_{T,t} = \dot{\theta}_{T,t} \cdot \Delta t \tag{4.13}$$

$$\Delta \phi_{T,t} = \dot{\phi}_{T,t} \cdot \Delta t \tag{4.14}$$

The model assumes that δV_T , $\delta \theta_T$, and $\delta \phi_T$ are constant since acceleration is held constant. Thus, the changes in velocity and heading must be constant as well. This is reflected in Equation 4.15 through 4.17 which define \dot{V} , $\dot{\theta}$, and $\dot{\phi}$.

$$\dot{V}_{T,t} = \delta V_{T,t-1} \Delta t \quad (4.15)$$

$$\dot{\theta}_{T,t} = \delta \theta_{T,t-1} \Delta t \quad (4.16)$$

$$\dot{\phi}_{T,t} = \delta \phi_{T,t-1} \Delta t \quad (4.17)$$

4.2.5 Camera Frame Equations.

As discussed earlier, the resulting states must be first be transformed to the camera coordinate frame using the rotation matrices specified in Equation 4.1. Both the target centroid and measurement points were transformed in this fashion. These transformations are described by Equation 4.18 and 4.19.

$$R_{DCM}^T \begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} = \begin{bmatrix} u_{Tc} \\ v_{Tc} \\ w_{Tc} \end{bmatrix} \quad (4.18)$$

$$R_{DCM}^T \begin{bmatrix} \dot{x}_T \\ \dot{y}_T \\ \dot{z}_T \end{bmatrix} = \begin{bmatrix} \dot{u}_{Tc} \\ \dot{v}_{Tc} \\ \dot{w}_{Tc} \end{bmatrix} \quad (4.19)$$

4.2.6 Pixel Frame Equations.

The final transformation is to the pixel coordinate frame so that the filter predicted measurements may be compared to actual measurements. The transformation results in a conversion from the three dimensional camera coordinate frame to the two dimensional pixel frame that is representative of what the camera would actually see. The resulting states are illustrated by Equation 4.20.

$$\begin{array}{llll} u_{Tc}, w_{Tc} & \rightarrow & u_{Tp} & \dot{u}_{Tc} \rightarrow u_{Tp} \\ v_{Tc}, w_{Tc} & \rightarrow & v_{Tp} & \dot{v}_{Tc} \rightarrow v_{Tp} \\ & & & \dot{w}_{Tc} \rightarrow s_T \end{array} \quad (4.20)$$

The measurement states for the target, u_{Tp} , v_{Tp} , s_T , \dot{u}_{Tp} , and \dot{v}_{Tp} are defined by Equation 4.21 through 4.25, where fl is the focal length of the camera).

$$u_{Tp} = \frac{u_{Tc}}{w_{Tc}} \cdot fl \quad (4.21)$$

$$v_{Tp} = \frac{v_{Tc}}{w_{Tc}} \cdot fl \quad (4.22)$$

$$s_T = \frac{1}{2} \cdot \left(\frac{-u_{Tc} \cdot \dot{w}_{Tc}}{w_{Tc}^2} + \frac{-v_{Tc} \cdot \dot{w}_{Tc}}{w_{Tc}^2} \right) \quad (4.23)$$

$$\dot{u}_{Tp} = \frac{\dot{u}_{Tc}}{w_{Tc}} \cdot fl \quad (4.24)$$

$$\dot{v}_{Tp} = \frac{\dot{v}_{Tc}}{w_{Tc}} \cdot fl \quad (4.25)$$

The focal length fl adds a dimension to an otherwise unitless measurement, the pixel. The focal length also determines the camera's angle of view. Longer focal lengths correspond in smaller view angles while shorter focal lengths correspond with larger view angles, as seen in Figure 4.6 [9]. Additional properties are provided in Table 4.2.

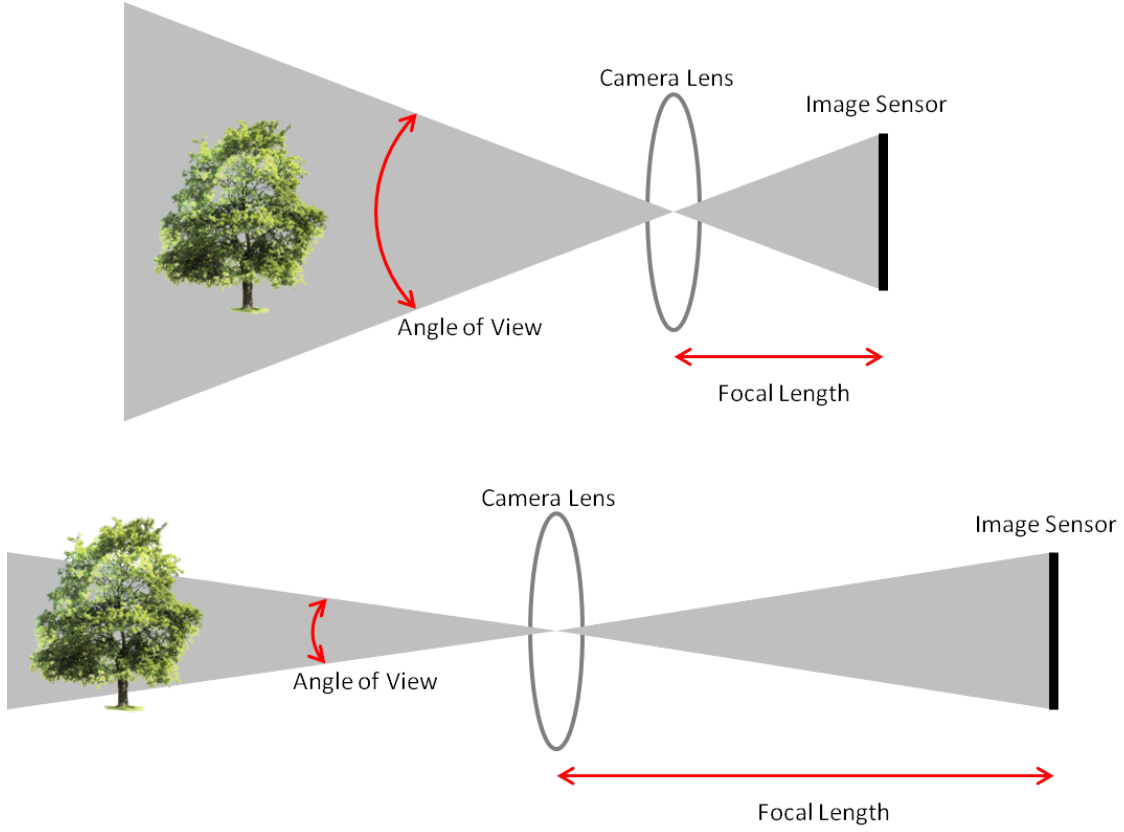


Figure 4.6: Relationship Between Focal Length and View Angle

Table 4.2: Image Properties

Focal Length	View Angle	Area Captured	Apparent Size
Short	Wide	Large	Small
Long	Small	Small	Large

The focal length also allows the calculation of angle error. Given a focal length, f , and the difference between the filter estimated location of the target, u_{fp} and actual location of the target, u_{Tp} , an angular error γ can be determined. This provides another useful metric

to compare the particle filter's performance across different systems.

$$\arctan\left(\frac{u_{Tp} - u_{fp}}{fl}\right) = \gamma \quad (4.26)$$

4.2.7 Acceleration and Depth Knowledge.

Since the pixel measurements, u_p , v_p , w , \dot{u}_p , and \dot{v}_p , are based on angle measurements, position estimates, such as depth, are impossible to determine without knowledge of an initial length scale. However, if the target's acceleration is non-zero and known, then the length scale can be determined by integrating acceleration to determine velocity and then by integrating velocity to determine acceleration. If the known acceleration is zero, no length scale can be determined since integrating results in a trivial solution. The results for EPF-B contained in 5 demonstrate the validity of requiring a non-zero acceleration.

4.3 Evaluated Particle Filter A

EPF-A uses the camera measurements, u_c , v_c , and w_c , to track the target. This filter simulates the data that might be generated by two cameras, or a system with a range finding device such as LIDAR. Additionally, EPF-A will characterize the performance of the particle filter tracking multiple hidden states in three dimensions. The hidden states are those that cannot be directly observed. For EPF-A, the hidden states are V , θ , and ϕ . The steps taken by the filter correspond to those in Section 2.5.5 and are subsequently discussed in specific detail.

4.3.1 Initialization and Proposal Distribution.

EPF-A uses the same model as the target to both generate the initial and subsequent proposal distributions and rotate to the camera frame. The key difference is that at the re-sampling step, the filter adds the system noise variance to generate the proposal distribution. Equation 4.27 through 4.29 show the states and manner in which variety is added at the sampling step. r_u is a random number sampled from a standard uniform distribution between 0 and 1. However, any distribution may be used to generate the variety; the

distribution is not required to be a white Gaussian distribution.

$$V_{f,t} = V_{f,t-1} + \Delta V_{f,t-1} + \sqrt{\sigma^2(x_{f,v})} \cdot r_u \quad (4.27)$$

$$\theta_{f,t} = \theta_{f,t-1} + \Delta \theta_{f,t-1} + \sqrt{\sigma^2(x_{f,\theta})} \cdot r_u \quad (4.28)$$

$$\phi_{f,t} = \phi_{f,t-1} + \Delta \phi_{f,t-1} + \sqrt{\sigma^2(x_{f,\phi})} \cdot r_u \quad (4.29)$$

Variety was only added to V_f , θ_f , and ϕ_f since the positions are derived from these states and it is the positions that are compared against the measurements once transformed into the appropriate frame. If variety were added directly to the states x_f , y_f , and z_f , this would not only be redundant, but also adversely affect the determination of the hidden states since the particle filter may select measurements that no longer directly corresponding to the hidden states that generated them. The necessary rotations are then performed to produce the camera frame position states, $u_{C,f}$, $v_{C,f}$, and $w_{C,f}$, for these particles.

4.3.2 Importance Sampling.

As discussed in Section 2.5.5, the particle filter must sample importance weights from the relative posterior PDF $p(\mathbf{y}_l | \tilde{\mathbf{x}}_l^{(i)})$. EPF-A uses the same distribution model as PPF-B, that is a normal distribution, discussed in Section 3.4.3 and detailed in Equation 3.15. Thus, the variables that must be set are the weighing matrix, W , and the noise variance in the measurement, $\sigma^2(\mathbf{m})$. Models containing multiple measurements can utilize a weighing matrix when calculating the importance weight for each particle. This allows for the model to account for the quality of each measurements; measurements that are known to be noisier than others may be assigned a lower weight within the weighing matrix and vice-versa. Equation 4.30 defines the weighing matrix used by EPF-A.

$$\Omega_{EPF-A} = \begin{bmatrix} \Omega(u) & 0 & 0 \\ 0 & \Omega(v) & 0 \\ 0 & 0 & \Omega(w) \end{bmatrix} \quad (4.30)$$

The importance weights are subsequently normalized using Equation 2.25.

4.3.3 Resampling and Global State Estimation.

EPF-A uses the re-sampling algorithm outlined in Section 2.5.5.4. The re-sampling step generates the approximation of the posterior filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. The estimated global states are determined by taking an average mean of the posterior filtering distribution.

4.3.4 Preliminary Evaluation.

Before conducting a battery of evaluation tests, a single test was performed to ensure EPF-A is able to function. Initial conditions for the target and filter differ, forcing the filter to locate and track the target, and measurement noise was added. Due to confidence in the performance of this filter, the target and filter used differing initial conditions to determine if the filter could track to the target. The filter is compared against the Simple Linear Model A (SLMA) and Evaluation Kalman Filter A (EKF-A), both comparison filters developed in Section 5.2.2.1 and 5.2.2.3 respectively. Additional tests and performance criteria, including hidden state evaluation, is discussed in Chapter 5. Table 4.3 and 4.4 contains the initial conditions and measurement variances. The results are plotted in Figure 4.7 and 4.8.

Table 4.3: Filter A Preliminary Evaluation Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
Target	5	5	5	1	45	45	0	0	0	0.1	5	5
Filter	0	0	10	0	0	0	0	0	0	0	0	0
Filter Variance	1	1	1	1	0.1	0.1	0.5	0.5	0.5	0.1	0.01	0.01

Table 4.4: Filter A Preliminary Evaluation Measurement Noise Variances

States	u	v	w
Target	5	5	5
Filter	5	5	5

As seen in Figure 4.7, the filter appears to track the target successfully according to the metrics developed in Section 5.2.1. Additional tests and performance criteria, including hidden state evaluation, is discussed in Chapter 5. Figure 4.8 demonstrates EPF-A’s ability to track the hidden states more accurately than either SLMA or EKF-A. Table 4.5 provides the mean values for the last 50 time steps.

Table 4.5: Filter A Preliminary Evaluation: Mean Errors

Metrics	x	y	z	V	θ	ϕ
SLMA	4.1725	4.2698	4.0238	97.3539	44.7735	121.9448
EKF-A	2.5025	6.1966	1.9997	37.7595	88.7395	56.9636
EPF-B	1.6095	1.2216	1.7188	1.9931	16.6655	51.0877

4.4 Evaluated Particle Filter B

EPF-B is nearly identical to EPF-A except that it uses the pixel measurements as measurements; that is the pixel location, u_{fp} and v_{fp} , and the velocities, s_f , \dot{u}_{fp} , and \dot{v}_{fp} , all defined by Equation 4.33 through 4.37. These pixel measurements are based on the pixel location of the target centroid. In contrast, the camera pixel measurements are eight sets of pixel measurements, one for each of the observed corner points of the target. These eight camera pixel measurements must be condensed to a single set of camera measurements that describe the centroid. The averages of the eight pixel positions, $\sum_{m=1}^M u_{Tp,t-1}^m$ and

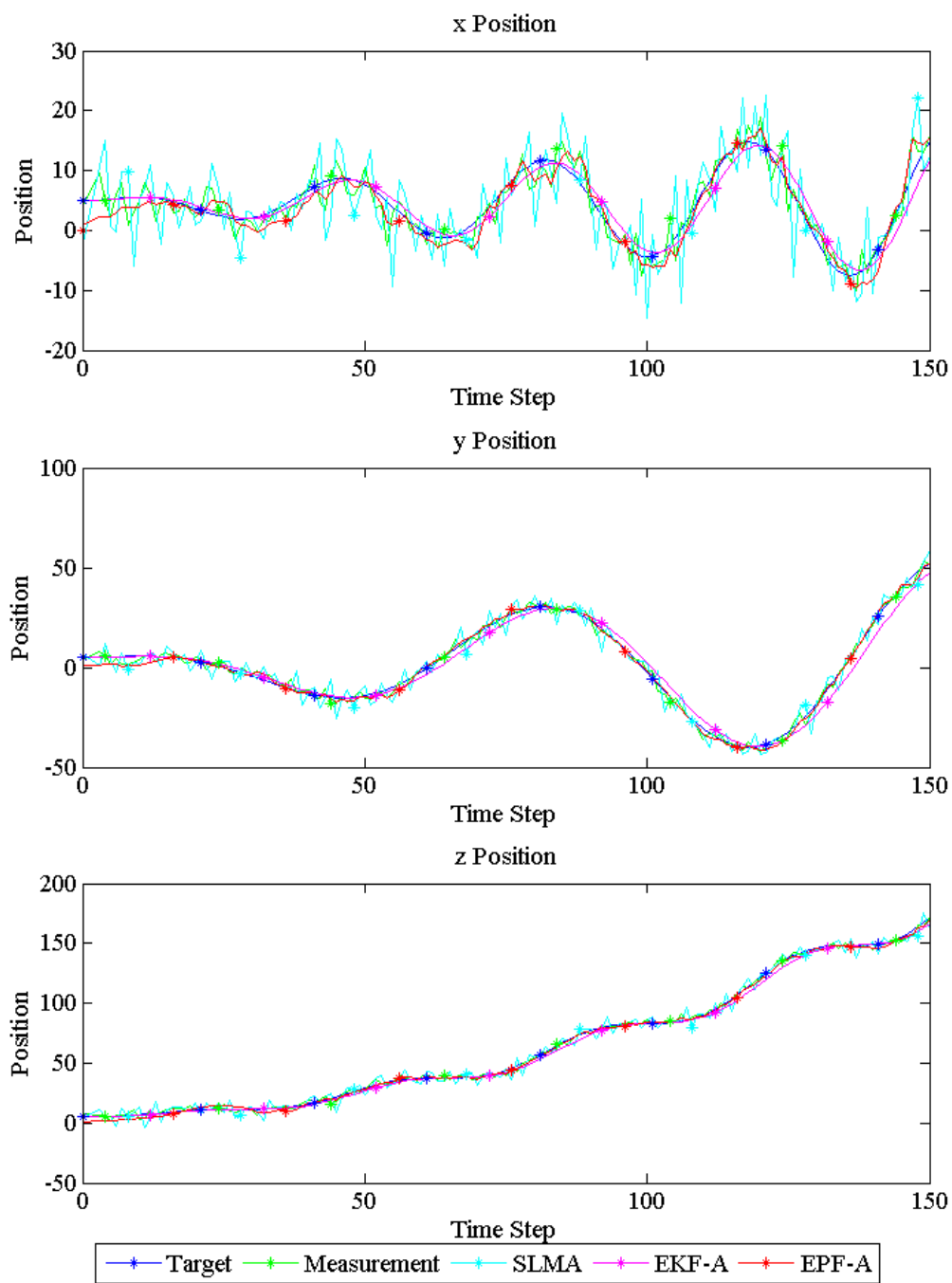


Figure 4.7: Filter A Preliminary Evaluation: Position

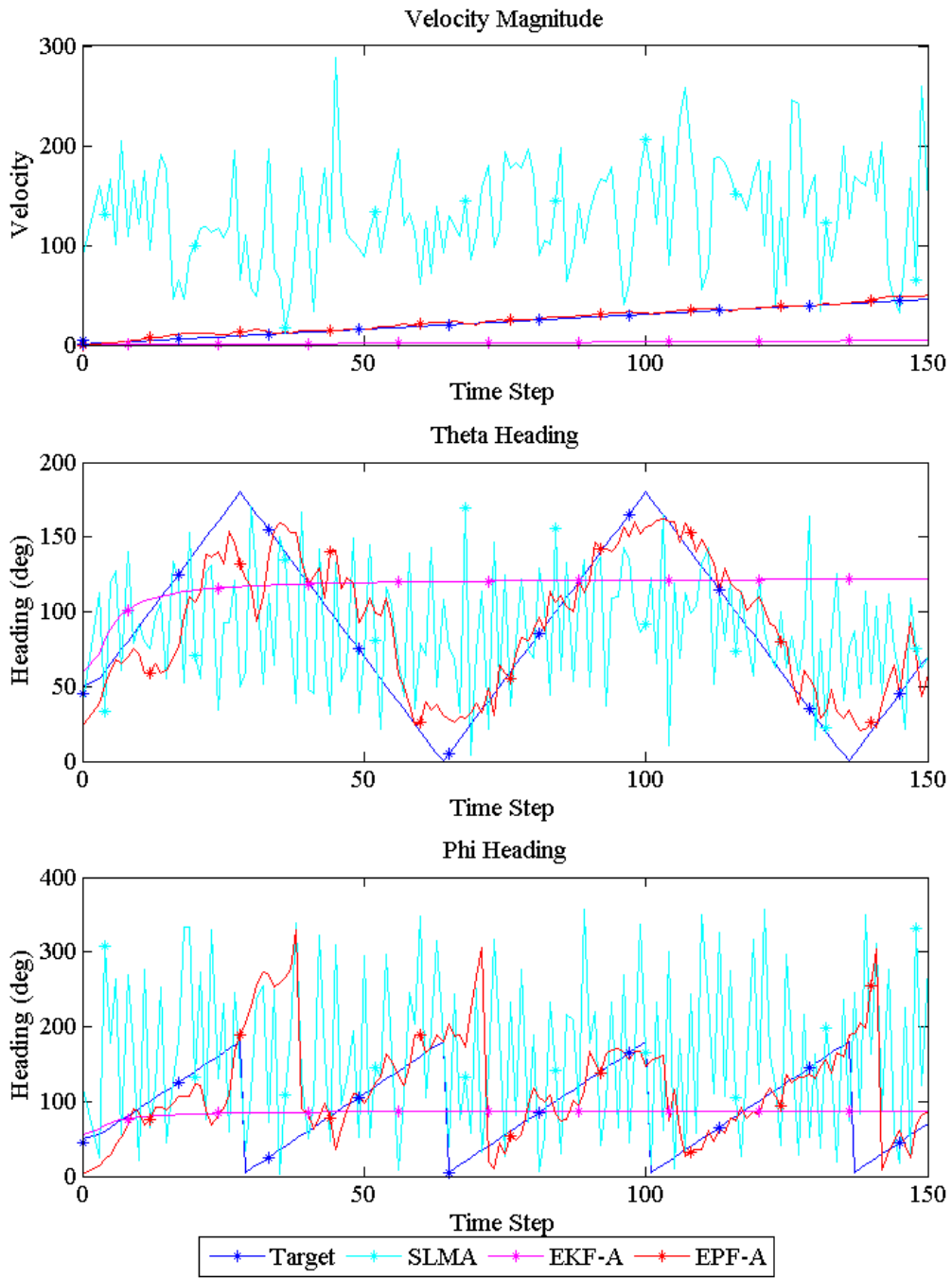


Figure 4.8: Filter A Preliminary Evaluation: Velocity

$\sum_{m=1}^M v_{Tp,t-1}^m$, are assumed to provide the representative values of the centroid location which are used as the measurement states u_p and v_p . Least mean squares determined the measurement states $s_T, \dot{u}_T p, \dot{v}_T p$, at each time step t , using an M number of matched points. Equation 4.31 and 4.32 detail the steps taken to determine the measurement states.

$$\begin{bmatrix} s_t & \dot{u}_{p,t} & \dot{v}_{p,t} \end{bmatrix} \begin{bmatrix} u_{Tp,t-1}^1 \dots u_{Tp,t-1}^M & v_{Tp,t-1}^1 \dots v_{Tp,t-1}^M \\ 0 \dots 0 & \Delta t \dots \Delta t \\ \Delta t \dots \Delta t & 0 \dots 0 \end{bmatrix} = \begin{bmatrix} \Delta u_{Tp,t}^1 \dots \Delta u_{Tp,t}^M & \Delta v_{Tp,t}^1 \dots \Delta v_{Tp,t}^M \end{bmatrix}$$

Dimensions	[1 × 3]	[3 × 2M]	[1 × 2M]
Labels	A	x	b

(4.31)

$$A = b \cdot x^\top \cdot (x \cdot x^\top)^{-1} \quad (4.32)$$

Thus, EPF-B, mirroring the target model, predicts the pixel measurements expected for each particle using Equation 4.33 through 4.37.

$$u_{p,f} = \frac{u_{C,f}}{w_{C,f}} \cdot fl \quad (4.33)$$

$$v_{p,f} = \frac{v_{C,f}}{w_{C,f}} \cdot fl \quad (4.34)$$

$$s_f = \frac{\dot{w}_{C,f}}{w_{fc}} \cdot fl \quad (4.35)$$

$$\dot{u}_{p,f} = \frac{\dot{u}_{C,f}}{w_{C,f}} \cdot fl \quad (4.36)$$

$$\dot{v}_{p,f} = \frac{\dot{v}_{C,f}}{w_{C,f}} \cdot fl \quad (4.37)$$

Additionally, the weighting matrix changes slightly as seen in Equation 4.38.

$$\Omega_{EPF-B} = \begin{bmatrix} \Omega(u_{p,f}) & 0 & 0 & 0 & 0 \\ 0 & \Omega(v_{p,f}) & 0 & 0 & 0 \\ 0 & 0 & \Omega(s_f) & 0 & 0 \\ 0 & 0 & 0 & \Omega(\dot{u}_{p,f}) & 0 \\ 0 & 0 & 0 & 0 & \Omega(\dot{v}_{p,f}) \end{bmatrix} \quad (4.38)$$

4.4.1 Preliminary Evaluation.

As with EPF-A, EPF-B underwent a preliminary test to evaluate its functionality. The first evaluation was movement along the x-axis at constant velocity, with no measurement noise, and the following initial conditions, seen in Table 4.6.

Table 4.6: Filter B Preliminary Evaluation Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
Target	0	0	5	1	90	0	0	0	0	0	0	0
Filter	0	0	0	0	0	0	0	0	0	0	0	0
Filter Variance	0.1	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1

The simulation used 500 particles and a measurement noise covariance of 30. Within 5 time steps, EPF-B crashed due to weight collapse. The last sum of weights was equal to 3.307×10^{-220} . Additional simulation runs resulted in similar results. Weight collapse, as discussed in 2.5.6, can be mitigated, typically with the addition of particles. Multiple subsequent evaluation were performed, using 1500 particles, and these too resulted in weight collapse. Figure 4.9 illustrates particle collapse from the perspective particle distribution.

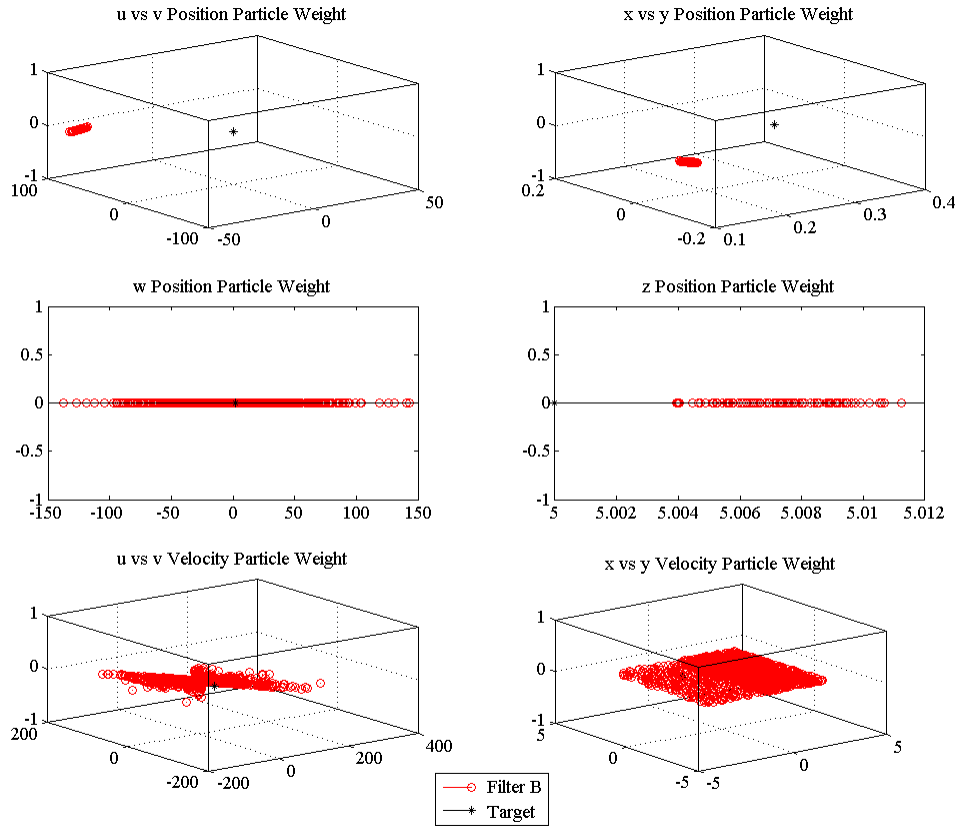


Figure 4.9: Particle Filter B Weight Collapse

Although there still exists a spread of particles, none have sufficient weight to be sampled. Part of this may stem from insufficient variance. As seen in the first subplot, entitled U vs V Position Particle Weight, the target value lies outside the spread of particles and hence, cannot be sampled. The subsequent sections detail the methods taken that attempt to mitigate weight collapse.

4.5 Weight Collapse Mitigation

One of the core concepts of the particle filter is variance and the spread of particles throughout the state space to include the target solution within the particle spread. As demonstrated in Section 4.4.1, the present form of EPF-B does not include the solutions

within its particle spread. In order to encompass the solution, variance must be in such a manner that expands the particle spread in the direction of the likely solution and not in a manner that simply adds more variation in all directions. Two methods were explored; variation along the depth vector, referred to as the Depth-Compensated Evaluation Particle Filter B (DC), and variation dependent on the relationship between in hidden and observable states, referred to as the Jacobian-Compensated Evaluation Particle Filter B (JC). One further change was also made to the weighing matrix for the importance weights.

4.5.1 Depth Vector Variation.

The DC is based on the unit vector of the target centroid as observed from the camera. Since the camera cannot detect depth, the $u_{C,f}$ and $v_{C,f}$ variables vary in proportion to $w_{C,f}$ with the same $u_{p,f} \times v_{p,f}$. The variance factors are based on s (the measurement of depth over time) while maintaining good measurements of heading. One of the drawbacks of this variation method is that it introduces variation into the position states. The position states are derived from the hidden variables and by introducing variation directly into the position states, the DC will affect the accuracy of the hidden states since a particle will be selected based on combination the hidden state's values and this injected variation.

4.5.2 State Jacobian Variation.

The JC avoids the adverse influence on the position states by only introducing variation into the hidden states directly. The JC attempts to determine the appropriate variance values, $\sigma^2(x_{f,v})$, $\sigma(x_{f,\theta})$, $\sigma^2(x_{f,\phi})$, for each time step based on the amount of influence the hidden states, V_f , θ_f , and ϕ_f , will have on the measurements s_f , $\dot{u}_{p,f}$, and $\dot{v}_{p,f}$. The reason these measurements were chosen is that they are the least certain and they are exclusively derived from the three hidden states mentioned. A first order approximation of Equation 4.6 through 4.8 is used to evaluate the sensitivity of the measurements to the

states by Equation 4.39, 4.40, and 4.41.

$$\dot{x}_{f,t} = V_{f,t} \cdot \sin(\theta_{f,t}) \cdot \sin(\phi_{f,t}) \cdot \Delta t \quad (4.39)$$

$$\dot{y}_{f,t} = V_{f,t} \cdot \cos(\theta_{f,t}) \cdot \Delta t \quad (4.40)$$

$$\dot{z}_f = V \cdot \sin(\theta_{f,t}) \cdot \sin(\phi_{f,t}) \cdot \Delta t \quad (4.41)$$

The velocity approximations are transformed to the camera oriented coordinates by Equation 4.18. The influence of the hidden states on the measurements can be determined by examining the eigenvalues and eigenvectors that describe the correlation between these variables. The general formula describing how the change in states, ΔX , affects the change in measurement states, ΔS , is described by Equation 4.42.

$$\Delta S = \frac{\partial S}{\partial X} \Delta X \quad (4.42)$$

The eigenvalues and eigenvectors are drawn from the $\frac{\partial S}{\partial X}$. Since the hidden states pass through several transformations within EPF-B before being used to generate measurements, a combination of Jacobian and rotation matrices are used, described and expanded in equations 4.43, 4.44, and 4.45. All states are based on the individual particle state estimates.

$$\begin{bmatrix} \Delta s \\ \Delta \dot{u}_p \\ \Delta \dot{v}_p \end{bmatrix} \approx \begin{bmatrix} \nabla_{\dot{u}, \dot{v}, \dot{w}} \begin{bmatrix} s \\ u_p \\ v_p \end{bmatrix} \end{bmatrix} [R_{G,C}] \begin{bmatrix} \nabla_{V, \theta, \phi} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \Delta V \\ \Delta \theta \\ \Delta \phi \end{bmatrix} \quad (4.43)$$

$$\begin{bmatrix} \partial s \\ \partial \dot{u}_p \\ \partial \dot{v}_p \end{bmatrix} = \begin{bmatrix} \frac{\partial s}{\partial u} & \frac{\partial s}{\partial v} & \frac{\partial s}{\partial w} \\ \frac{\partial \dot{u}_p}{\partial u} & \frac{\partial \dot{u}_p}{\partial v} & \frac{\partial \dot{u}_p}{\partial w} \\ \frac{\partial \dot{v}_p}{\partial u} & \frac{\partial \dot{v}_p}{\partial v} & \frac{\partial \dot{v}_p}{\partial w} \end{bmatrix} [R_{G,C}] \begin{bmatrix} \frac{\partial \dot{x}}{\partial v} & \frac{\partial \dot{x}}{\partial \theta} & \frac{\partial \dot{x}}{\partial \phi} \\ \frac{\partial \dot{y}}{\partial v} & \frac{\partial \dot{y}}{\partial \theta} & \frac{\partial \dot{y}}{\partial \phi} \\ \frac{\partial \dot{z}}{\partial v} & \frac{\partial \dot{z}}{\partial \theta} & \frac{\partial \dot{z}}{\partial \phi} \end{bmatrix} \quad (4.44)$$

$$\begin{bmatrix} \partial s \\ \partial \dot{u}_p \\ \partial \dot{v}_p \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{w} \\ \frac{1}{w} & 0 & 0 \\ 0 & \frac{1}{w} & 0 \end{bmatrix} [R_{G,C}] \begin{bmatrix} \cos \theta \sin \phi \Delta t & -V \sin \theta \sin \phi \Delta t & V \cos \theta \cos \phi \Delta t \\ \cos \phi \Delta t & 0 & -V \sin \phi \Delta t \\ \sin \theta \sin \phi \Delta t & V \cos \theta \sin \phi \Delta t & V \cos \phi \sin \theta \Delta t \end{bmatrix} \quad (4.45)$$

The eigenvectors, \mathbf{V} , describe the direction and relative sensitivity of each factor. In other words if the measurement would be positively or negatively affected by each state and in what proportion. The eigenvalues, λ , describe the amount of influence for each eigenvector. The measurement state variances are determined using the eigenvectors and eigenvalues, along with scaling values, s_{f_1} and s_{f_2} . s_{f_1} scales the range and s_{f_2} sets the minimum variance.

$$\sigma^2(x_{V,\theta,\phi}) = s_{f_1} \cdot \mathbf{V} \cdot \text{diag}(\text{diag}(\lambda) + s_{f_2})^{-1} \quad (4.46)$$

4.5.3 *Weighing Matrix Adjustment.*

Of the five pixel measurements $u_{T,p}$, $v_{T,p}$, s_T , $\dot{u}_{T,p}$, and $\dot{v}_{T,p}$, the filter is least certain of s_T since it does not directly correspond to an actual depth variable. Instead, is an estimate dependent on $\dot{w}_{T,p}$ and $w_{T,p}$, defined in Equation 4.25. Since $\dot{w}_{T,p}$ is a measurement of velocity, the accuracy of the depth estimation might be increased by increasing the weight of the three velocity measurements, $\dot{u}_{T,p}$, $\dot{v}_{T,p}$, and s_T . The reasoning for this proposal is that by weighting the velocities higher, less weight will be placed on the pixel position since the pixel position is easier to measure. Weighting the velocities places an emphasis on depth to match the known acceleration, which is the only parameter that can provide an unambiguous estimate of range. This proposed adjustment will be evaluated in Section 5.4.

V. Simulation Tests and Results

5.1 Introduction

Once finalized, both EPF-A and EPF-B were evaluated using a variety of scenarios. Due to the differences in the filters, each underwent different simulations. All scenarios were conducted both without and with measurement noise to simulate realistic noisy measurements. Results from these simulations were analyzed using various metrics in order to evaluate the performance of both filters. Both filters were also contrasted against linear models, SLMA and Simple Linear Model B (SLMB) that used the same measurements provided to the respective filters. The details of these metrics and the accompanying linear comparison models, SLMA and SLMB, are discussed in Section 5.2.1 and 5.2.2 respectively.

5.2 Particle Filter Performance Evaluation

Performance metrics must be produced in order to evaluate the viability of particle filters. However, particle filters, due to their stochastic nature, pose two unique challenges compared to linear filters, such as the Kalman. One challenge is devising performance metrics. Due to their stochastic nature, each time the filter executes, the resulting estimates will be different given the same target model and measurements. In contrast, a linear filter will consistently return the same estimations each time it is executed, assuming the same measurements are provided each time. In order to develop a better comparison to a linear filter, either noise must be introduced to change the measurements or the states themselves must be changed either by introducing system noise or by changing the initial conditions. An additional challenge is devising a reasonable comparison to compare the filter against. Without a baseline comparison, it is difficult to quantify how well the filter is tracking the target. Although the filter may appear to estimate the target states, the question that must be

addressed is how much better does it estimate the target states compared to random guesses of the target states.

5.2.1 Performance Metrics.

Since each execution of the particle filter results in different predictions, any proposed metric must evaluate the filter's performance over multiple simulations. With this requirement in mind, one of the primary questions is how well does the filter predict the target states, in particular the hidden states. This question may also be posed as what is the difference, or error, between the target state and the filter estimate of that state. Two metrics were devised that answer this question in different manners: the Mean Absolute Error (MAE) and the Threshold Error Range (TER).

5.2.1.1 Mean Absolute Error.

The MAE provides a mean of absolute errors between the target states, \mathbf{x} , and filter estimate of those states, \mathbf{x}_f , over an q number of simulations at each time step, k .

$$\mathbf{x}_{MAE_t} = \frac{1}{q} \sum_i^q |\mathbf{x}_{f_i} - \mathbf{x}_i|_t \quad (5.1)$$

The MAE shows what values the filter could reasonably be expected to return, accounting for both good and poor estimations. Additionally, since the MAE is calculated for each time step, specific time ranges can be evaluated; for instance, the mean estimates when the filter is tracking to the target or how much variance could be expected between the different filters when each has acquired and is actively tracking to the target. A drawback of the MAE is that it does not determine the proportion of accurate estimations to poor estimations. For instance, a reasonable mean may be returned from several good estimates and a handful of dismal estimates. These dismal estimates, if numerous or severe enough, could invalidate the filter, but may be obscured by the mean and never noticed otherwise. The second metric, the TER attempts to address this issue by determining an effective operating range of the filter.

5.2.1.2 *Threshold Error Range.*

The TER first determines the absolute error between the target states and filter predictions, at each time step, for each simulation. At each time step, these error values are sorted from lowest, representing the smallest error, to highest, representing the greatest error. A percentile value, τ , determines the ratio of error values to return. If τ were 1, then the TER would always return the maximum error found from all simulations at each time step. If τ were 0.5, then the TER would return the error value for which half of the simulation errors were equal to or less than for each time step; in other words the median. Since the TER is not a mean, it is a better predictor of extreme behavior, hence the term threshold. If a τ of 0.9 is selected, then for the error returned at a particular time step, 90% of all simulations would have errors less than or equal to that value at that time step. The TER determines the error to return by first determining the appropriate index number by multiplying the selected percentile value, τ , by the number of simulations, q . The errors are sorted lowest to highest and the error at the index corresponding to the percentile value is returned. Equation 5.2 and 5.3 define the TER calculation, where `sort` is a function ranking the corresponding errors lowest to highest.

$$\mathbf{xs}_t \triangleq \text{sort} \left| \mathbf{x}_T - \mathbf{x}_f \right|_t \quad (5.2)$$

$$\mathbf{x}_{TER,t} = \mathbf{xs}_t (\lceil \tau \cdot q \rceil) \quad (5.3)$$

For all scenarios detailed in Chapter 5, a threshold of 0.9 was used, meaning 90% of all simulations in a particular scenario had errors less than or equal to the error value returned by TER. Figure 5.1 provides a visual depiction of the TER.

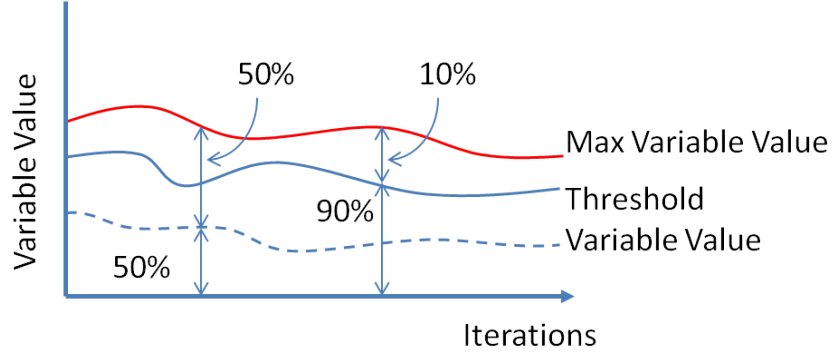


Figure 5.1: Threshold Error

5.2.1.3 Metric Variables.

The previous performance metrics, MAE and TER evaluated performance based on the following variables: D , the Euclidean distance between the target and filter prediction, V , the velocity magnitude, and the heading angles, θ and ϕ . D is defined by Equation 5.4. Figure 5.2 provides a visual depiction of the metric variables.

$$D_t = \sqrt{(x_{f,t} - x_{T,t})^2 + (y_{f,t} - y_{T,t})^2 + (z_{f,t} - z_{T,t})^2} \quad (5.4)$$

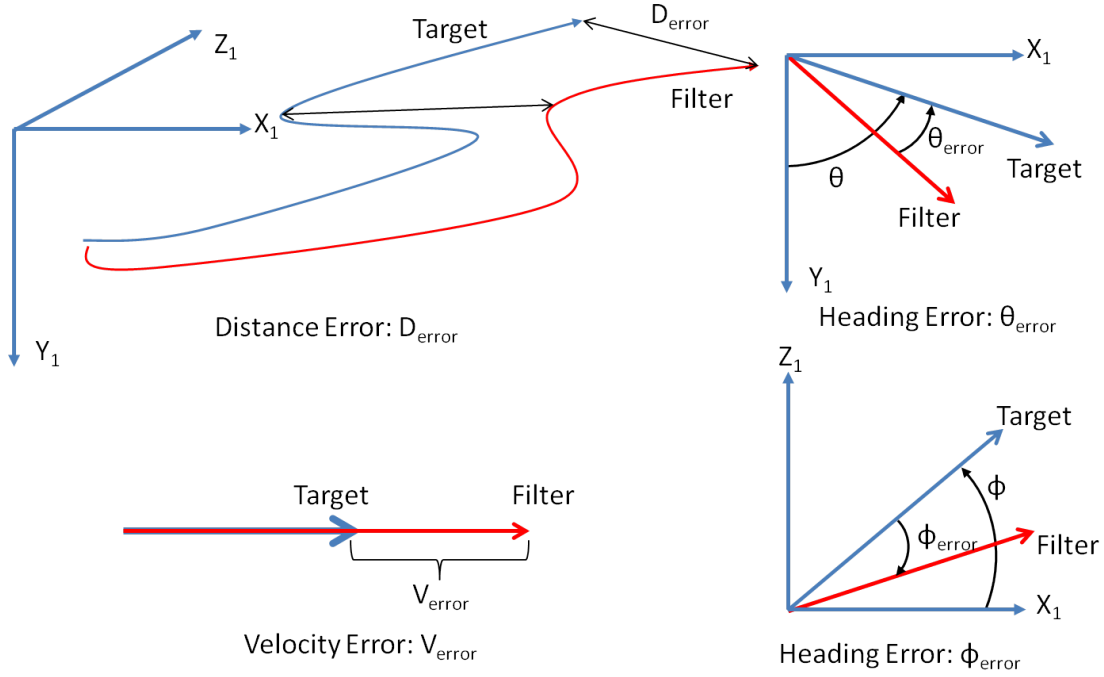


Figure 5.2: Performance Metric Variables

Both D and V have unlimited error ranges, while θ and ϕ are constrained between 0° and 180° since the greatest possible error is a heading in the opposite direction.

5.2.2 Comparison Models.

In order to provide reasonable comparisons for the particle filter, two linear models were developed, SLMA and SLMB, to serve as comparisons for EPF-A and EPF-B respectively. Each uses the same measurements provided to the filter it is compared against, and attempts to predict the target states. Additionally a Kalman filter, EKF-A similar to PKF-B, was developed as an additional comparison for EPF-A. A comparison Kalman filter was not developed for EPF-B due to EPF-B's immaturity compared to EPF-A as discussed in Section 4.4 and 5.4.

5.2.2.1 Linear Model A: Filter A.

The SLMA uses the measurements provided to EPF-A, which are u , v , and w , to predict the corresponding global values. Since only position is measured, velocity

is approximated using a first order backward difference approximation. The SLMA is modeled using Equation 5.5 through 5.6.

$$u_{LM,t} = u_{LM,t-1} + \dot{u}_{LM,t-1}\Delta t \quad (5.5)$$

$$v_{LM,t} = v_{LM,t-1} + \dot{v}_{LM,t-1}\Delta t \quad (5.6)$$

$$w_{LM,t} = w_{LM,t-1} + \dot{w}_{LM,t-1}\Delta t \quad (5.7)$$

The variables are then rotated from the camera frame to the global frame using the DCM from Equation 4.1. The variable transformations are provided by Equation 5.8.

$$(R_{DCM_t}) \begin{bmatrix} u_{LM,t} \\ v_{LM,t} \\ w_{LM,t} \end{bmatrix} \rightarrow \begin{bmatrix} x_{LM,t} \\ y_{LM,t} \\ z_{LM,t} \end{bmatrix} \quad (5.8)$$

The component velocities, $\dot{x}_{LM,t}$, $\dot{y}_{LM,t}$, and $\dot{z}_{LM,t}$ are determined using Equation 5.9 through 5.11.

$$\dot{x}_{LM,t} = \frac{x_{LM,t} - x_{LM,t-1}}{\Delta t} \quad (5.9)$$

$$\dot{y}_{LM,t} = \frac{y_{LM,t} - y_{LM,t-1}}{\Delta t} \quad (5.10)$$

$$\dot{z}_{LM,t} = \frac{z_{LM,t} - z_{LM,t-1}}{\Delta t} \quad (5.11)$$

The hidden states, V and headings θ and ϕ can now be estimated with the estimated global state variables, using Equation 5.12, 5.13, and 5.14.

$$V_{LM,t} = \sqrt{\dot{x}_{LM,t}^2 + \dot{y}_{LM,t}^2 + \dot{z}_{LM,t}^2} \quad (5.12)$$

$$\theta_{LM,t} = \arccos\left(\frac{\dot{y}_{LM,t}}{V_{LM,t}}\right) \quad (5.13)$$

$$\phi_{LM,t} = \arctan\left(\frac{\dot{z}_{LM,t}}{\dot{x}_{LM,t}}\right) \quad (5.14)$$

5.2.2.2 Linear Model B: Filter B.

The SLMB is similar to SLMA regarding the calculation of estimated states, however SLMB uses the measurements u_p , v_p , s , \dot{u}_p , and \dot{v}_p , along with the target value for \dot{w}_{tgt_k} .

At least one target variable must be used since the SLMB attempts to extract 3-D variables from Two-Dimensional (2-D). The use of this target variable is minimized to prevent the SLMB having too significant an advantage over EPF-B.

$$u_{LM,t} = \frac{w_{LM,t-1} \cdot u_{LMp,t}}{f} \quad (5.15)$$

$$v_{LM,t} = \frac{w_{LM,t-1} \cdot v_{LMp,t}}{f} \quad (5.16)$$

$$w_{LM,t} = \sqrt{\frac{\dot{w}_{T,t} \cdot (-u_{LM,t} - v_{LM,t})}{2 \cdot s}} \quad (5.17)$$

$$\dot{u}_{LM,t} = \frac{\dot{u}_{LMp,t} \cdot w_{LM,t}}{f} \quad (5.18)$$

$$\dot{v}_{LM,t} = \frac{\dot{v}_{LMp,t} \cdot w_{LM,t}}{f} \quad (5.19)$$

$$\dot{w}_{LM,t} = \frac{w_{LM,t} - w_{LM,t-1}}{\Delta t} \quad (5.20)$$

These variables are then rotated from the camera frame to the global frame using the inverse DCM, defined in Equation 4.1. The transformations are the same as those defined in Equation 5.8. The hidden states, V and headings θ and ϕ can now be estimated with these rotated variables using Equation 5.12 through 5.14.

5.2.2.3 Evaluate Kalman Filter A.

The EKF-A is similar to PKF-A and PKF-B, both developed in Section 3.4.4, but EKF-A is expanded to 3-D space instead of 2-D space. The positional states are x , y , z , while the target velocity states are linearized from V , θ , and ϕ to the component velocities \dot{x} , \dot{y} , and \dot{z} as seen in Equation 5.21 through 5.23.

$$\dot{u}_{k,t} = V_{k,t} \cdot \sin(\theta_{k,t}) \cdot \cos(\phi_{k,t}) \cdot \Delta t \quad (5.21)$$

$$\dot{v}_{k,t} = V_{k,t} \cdot \cos(\theta_{k,t}) \cdot \Delta t \quad (5.22)$$

$$\dot{w}_{k,t} = V_{k,t} \cdot \sin(\theta_{k,t}) \cdot \sin(\phi_{k,t}) \cdot \Delta t \quad (5.23)$$

The component velocities are transformed back to velocity magnitude and heading via Equation 5.24 through 5.26 in order to provide a direct comparison to the target states.

$$\dot{u}_{k,t} = \sqrt{\dot{x}_{k,t}^2 + \dot{y}_{k,t}^2 + \dot{z}_{k,t}^2} \quad (5.24)$$

$$\dot{v}_{k,t} = \arccos\left(\frac{\dot{\theta}_{k,t}}{\dot{V}_{k,t}}\right) \quad (5.25)$$

$$\dot{w}_{k,t} = \arctan\left(\frac{\dot{\phi}_{k,t}}{\dot{V}_{k,t}}\right) \quad (5.26)$$

The control law, u_k , consists of the component accelerations, \ddot{x} , \ddot{y} , and \ddot{z} , respectively. The control law components are determined from the initial conditions by Equation 5.27 through 5.29.

$$u_{k,\ddot{x}} = \frac{\dot{V}}{\Delta t} \cdot \sin(\theta) \cdot \cos(\phi) \frac{\Delta t^2}{2} + \frac{\dot{\theta}}{\Delta t} \cdot V \cdot \cos(\theta) \cos(\phi) \frac{\Delta t^2}{2} - \frac{\dot{\phi}}{\Delta t} \cdot V \cdot \sin(\theta) \cdot \sin(\phi) \frac{\Delta t^2}{2} \quad (5.27)$$

$$u_{k,\ddot{y}} = \frac{\dot{V}}{\Delta t} \cdot \cos(\theta) \cdot \frac{\Delta t^2}{2} - \frac{\dot{\theta}}{\Delta t} \cdot V \cdot \sin(\theta) \frac{\Delta t^2}{2} \quad (5.28)$$

$$u_{k,\ddot{z}} = \frac{\dot{V}}{\Delta t} \cdot \sin(\theta) \cdot \sin(\phi) \frac{\Delta t^2}{2} + \frac{\dot{\theta}}{\Delta t} \cdot V \cdot \cos(\theta) \sin(\phi) \frac{\Delta t^2}{2} - \frac{\dot{\phi}}{\Delta t} \cdot V \cdot \sin(\theta) \cdot \cos(\phi) \frac{\Delta t^2}{2} \quad (5.29)$$

Similarly, the system noise covariance values, $\sigma x_{\dot{x},t}^2$, $\sigma x_{\dot{y},t}^2$, and $\sigma x_{\dot{z},t}^2$ are also derived from the target covariances using Equation 5.30 through 5.33.

$$\sigma x_{\dot{x},t}^2 = \sigma x_{V,t}^2 \cdot \sin(\sigma x_{\theta,t}^2) \cdot \cos(\sigma x_{\phi,t}^2) \cdot \Delta t \quad (5.30)$$

$$\sigma x_{\dot{y},t}^2 = \sigma x_{V,t}^2 \cdot \cos(\sigma x_{\theta,t}^2) \cdot \Delta t \quad (5.31)$$

$$\sigma x_{\dot{z},t}^2 = \sigma x_{V,t}^2 \cdot \sin(\sigma x_{\theta,t}^2) \cdot \sin(\sigma x_{\phi,t}^2) \cdot \Delta t \quad (5.32)$$

Equation 5.33 through 5.38 describe the the state transition model matrix, A , the control-input model, B , the control vector, u , the system noise covariance matrix, Q , the measurement model, C , and the measurement noise covariance matrix, R .

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.33)$$

$$\mathbf{B} = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \quad (5.34)$$

$$\mathbf{u}_k = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \quad (5.35)$$

$$\mathbf{Q} = \begin{bmatrix} \sigma(x_x) & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma(x_y) & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma(x_z) & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma(x_{\dot{x}}) & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma(x_{\dot{y}}) & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma(x_{\dot{z}}) \end{bmatrix} \quad (5.36)$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (5.37)$$

$$\mathbf{R} = \begin{bmatrix} \sigma(y_x) & 0 & 0 \\ 0 & \sigma(y_y) & 0 \\ 0 & 0 & \sigma(y_z) \end{bmatrix} \quad (5.38)$$

For all test cases within Chapter 5, EKF-A used the target initial conditions and accurate control law based on the target's initial conditions. PKF-B already demonstrated within Chapter 3 the inaccuracies of the Kalman filter when provided with inaccurate initial conditions or inaccurate control laws. Additionally, the measurements, u , v , and w were rotated using Equation 4.19.

5.3 Evaluated Particle Filter A

EPF-A simulates the measurements received by two cameras, u , v , and w , as well camera pan and tilt angles. Although this information would normally be obtained from two

cameras, only one camera rotation is used in order to simplify the modeling calculations and reduce potential unintended sources of error. Three types of scenarios were tested:

1. Movement about the orthogonal axes with constant velocity
2. Movement in a circle about the three principal axes with constant speed
3. Movement with constant acceleration for velocity and heading angles

Although the scenarios use different initial conditions, several of the parameters are constant, as defined in Table 5.1. A time step of 0.1 was chosen for the purposes of scaling.

Table 5.1: EPF-A Scenario Parameters

Parameter	EPF-A
Number of Simulations	100
Time Step	0.1
Number of Particles	500

5.3.1 Orthogonal Axes Movement.

Six scenarios were conducted, two about each axis, one with no measurement noise and one with measurement noise. All simulations ran for 150 time steps. Table 5.2 lists the initial conditions for each scenario and Table 5.3 lists the the variance when noise was used.

Table 5.2: Orthogonal Axes Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
x-Axis Movement												
Target	1	0	0	4	90	0	0	0	0	0	0	0
SLMA	1	0	0	4	90	0	0	0	0	0	0	0
EPF-A	0	0	0	0	0	0	0	0	0	0	0	0
y-Axis Movement												
Target	0	1	0	4	0	0	0	0	0	0	0	0
SLMA	0	1	0	4	0	0	0	0	0	0	0	0
EPF-A	0	1	0	0	0	0	0	0	0	0	0	0
z-Axis Movement												
Target:	0	0	10	4	90	90	0	0	0	0	0	0
SLMA	0	0	10	4	90	90	0	0	0	0	0	0
EPF-A	0	0	10	0	0	0	0	0	0	0	0	0
EPF-A Variance:	1	1	1	1	0.1	0.1	0.5	0.5	0.5	0.1	0.01	0.01

Table 5.3: Measurement Noise Variations

	No Noise			Noise		
Measurements	u	v	w	u	v	w
Target	0	0	0	5	5	5
EPF-A	0	0	0	5	5	5

The performance of each axial movement was evaluated using the performance metrics discussed in Section 5.2.1. Axial movements were first evaluated without measurement noise. Figure 5.3 contains the MAE values without noise for EPF-A and SLMA. Figure 5.4 contains the TER values without noise for EPF-A and SLMA. A single run for movement about each axis is shown in Figure 5.5, 5.6, and 5.7. The abbreviations used within the legend of each plot are detailed in Table 5.4.

Table 5.4: Evaluated Particle Filter A Legend Acronyms

Acronym	Meaning
P	EPF-A
L	SLMA
K	EKF-A
x	Axial movement along X axis
y	Axial movement along Y axis
z	Axial movement along Z axis
xy	Circular rotation in X/Y plane
xz	Circular rotation in X/Z plane
yz	Circular rotation in Y/Z plane

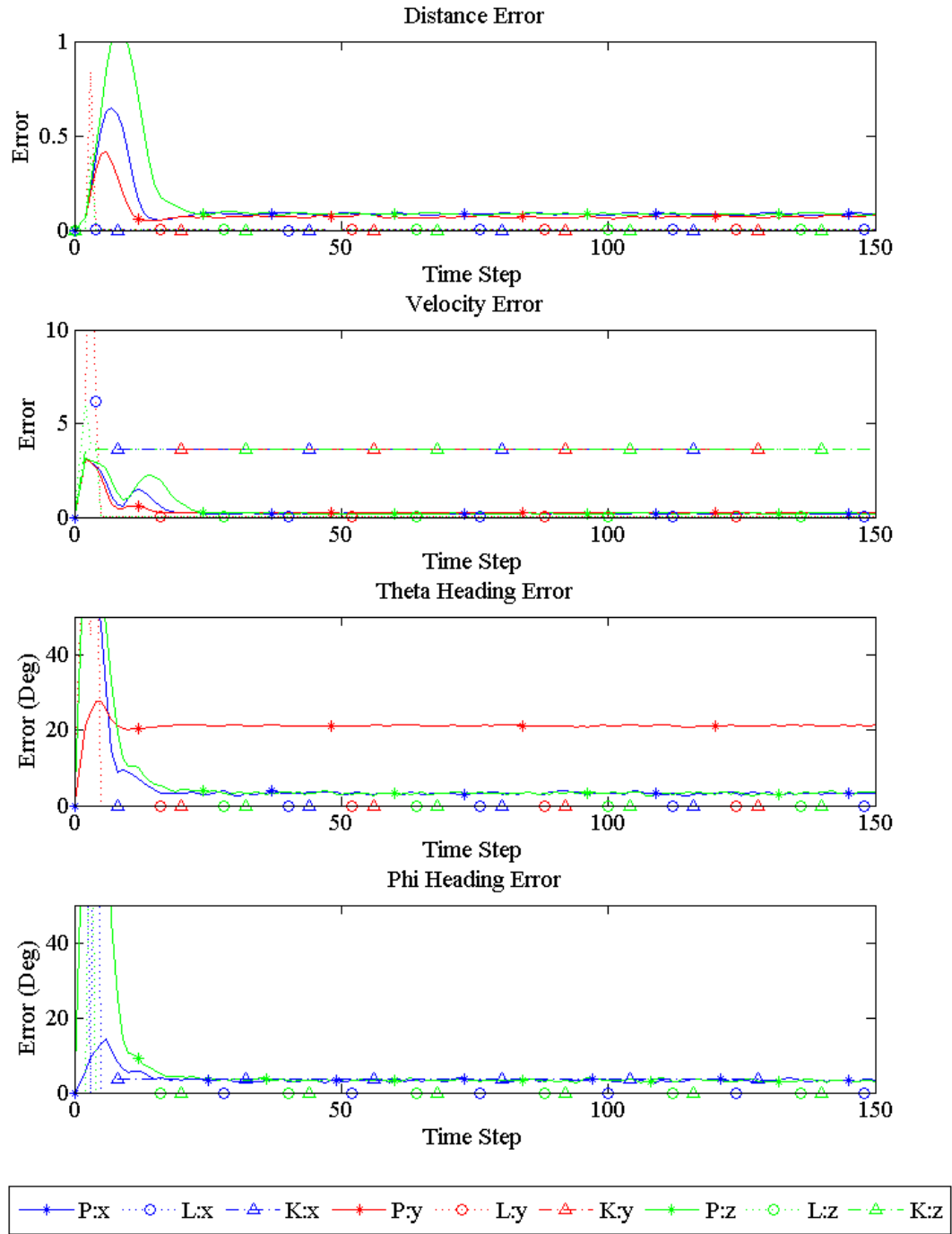


Figure 5.3: Mean Error for Axial Movement without Measurement Noise

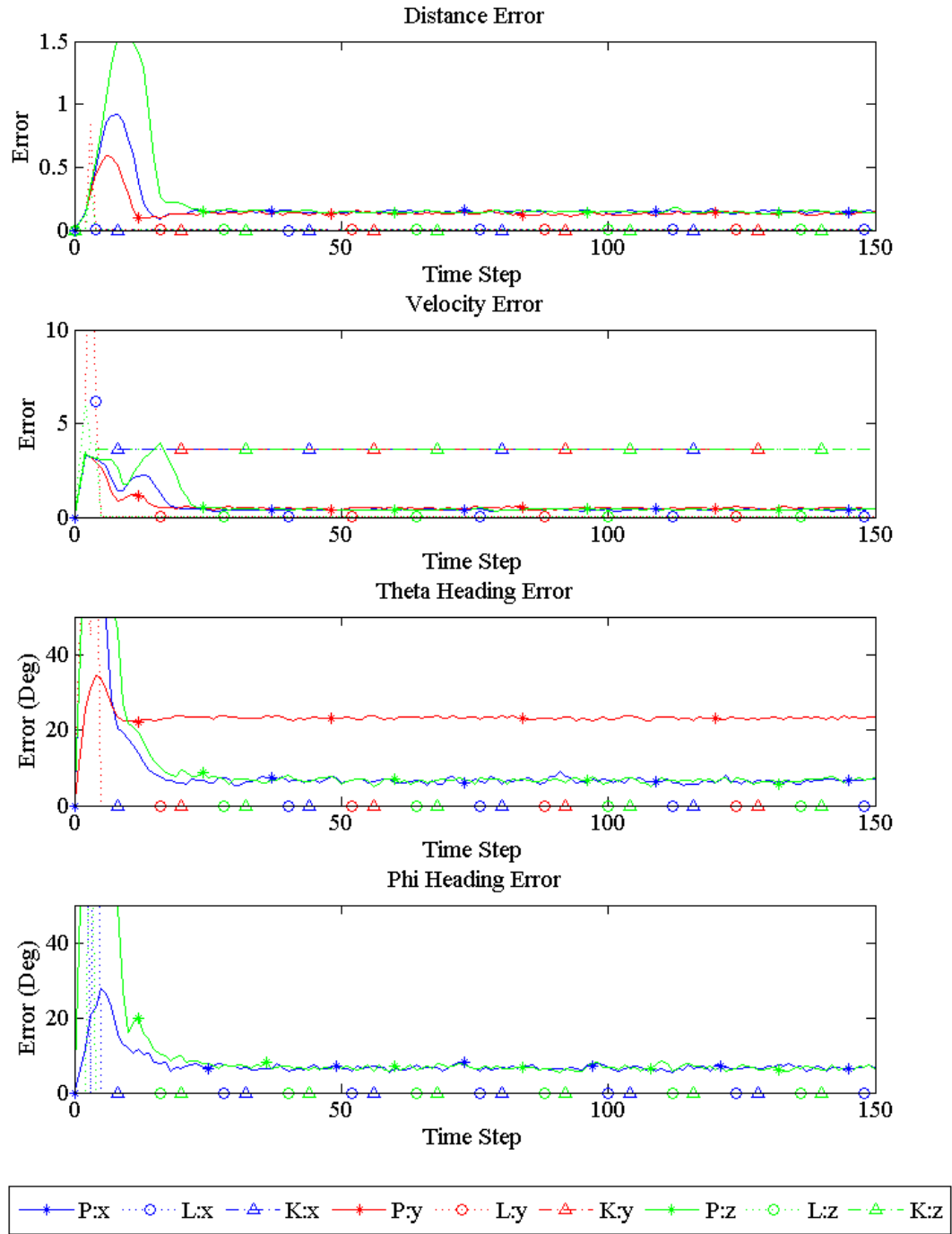


Figure 5.4: Threshold Error for Axial Movement without Measurement Noise

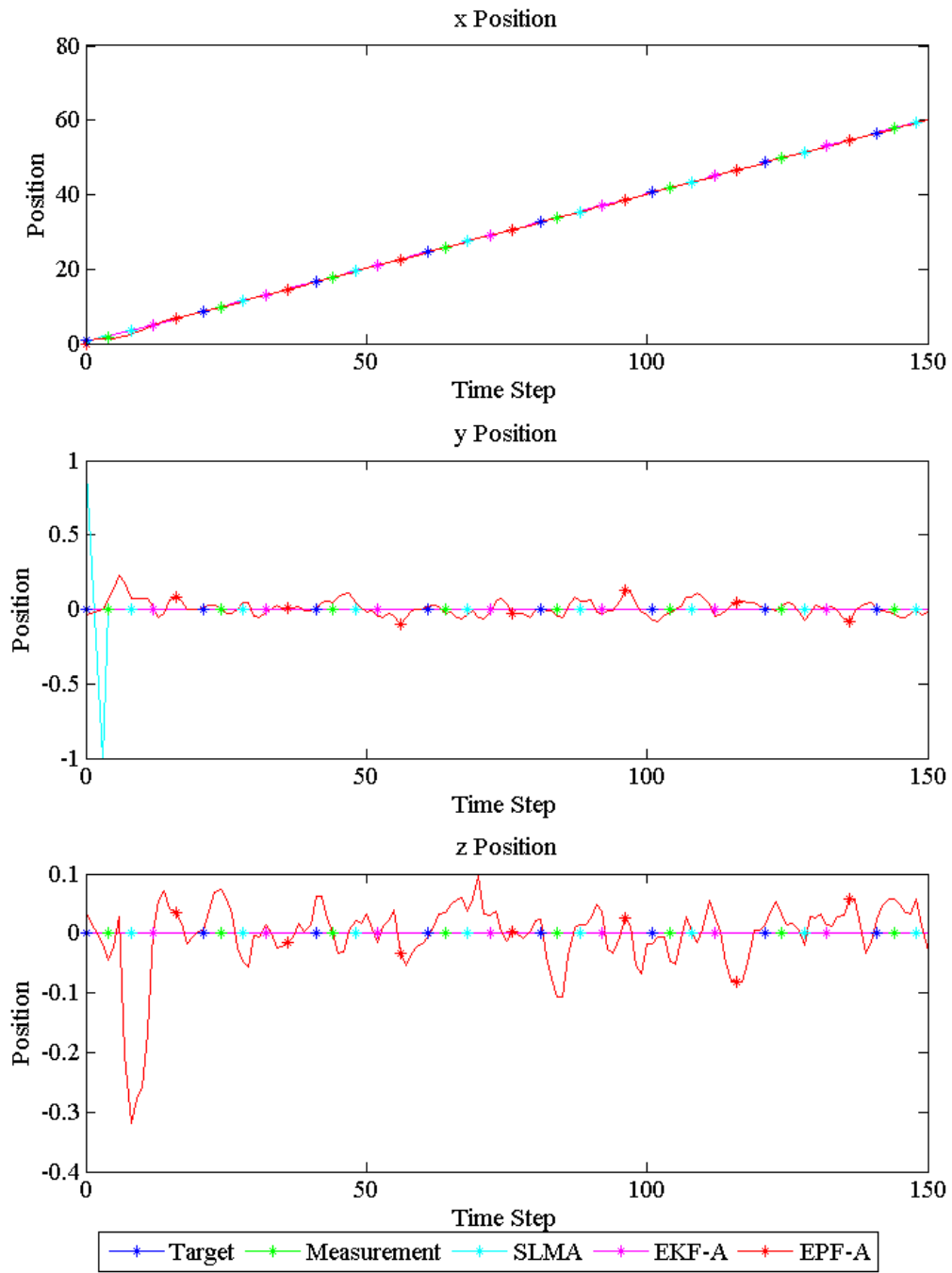


Figure 5.5: Single Simulation for x-Axis Movement without Measurement Noise

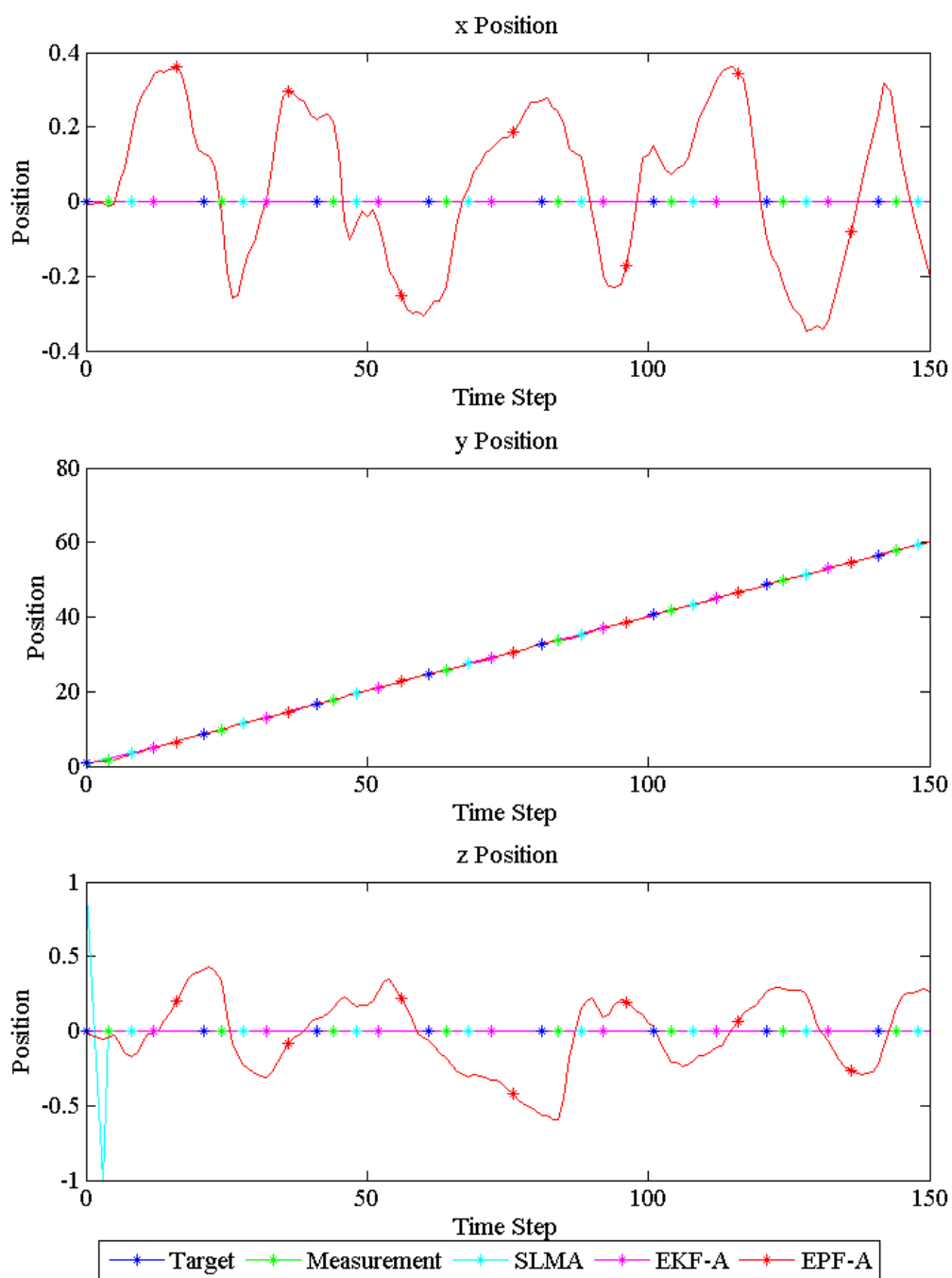


Figure 5.6: Single Simulation for y-Axis Movement without Measurement Noise

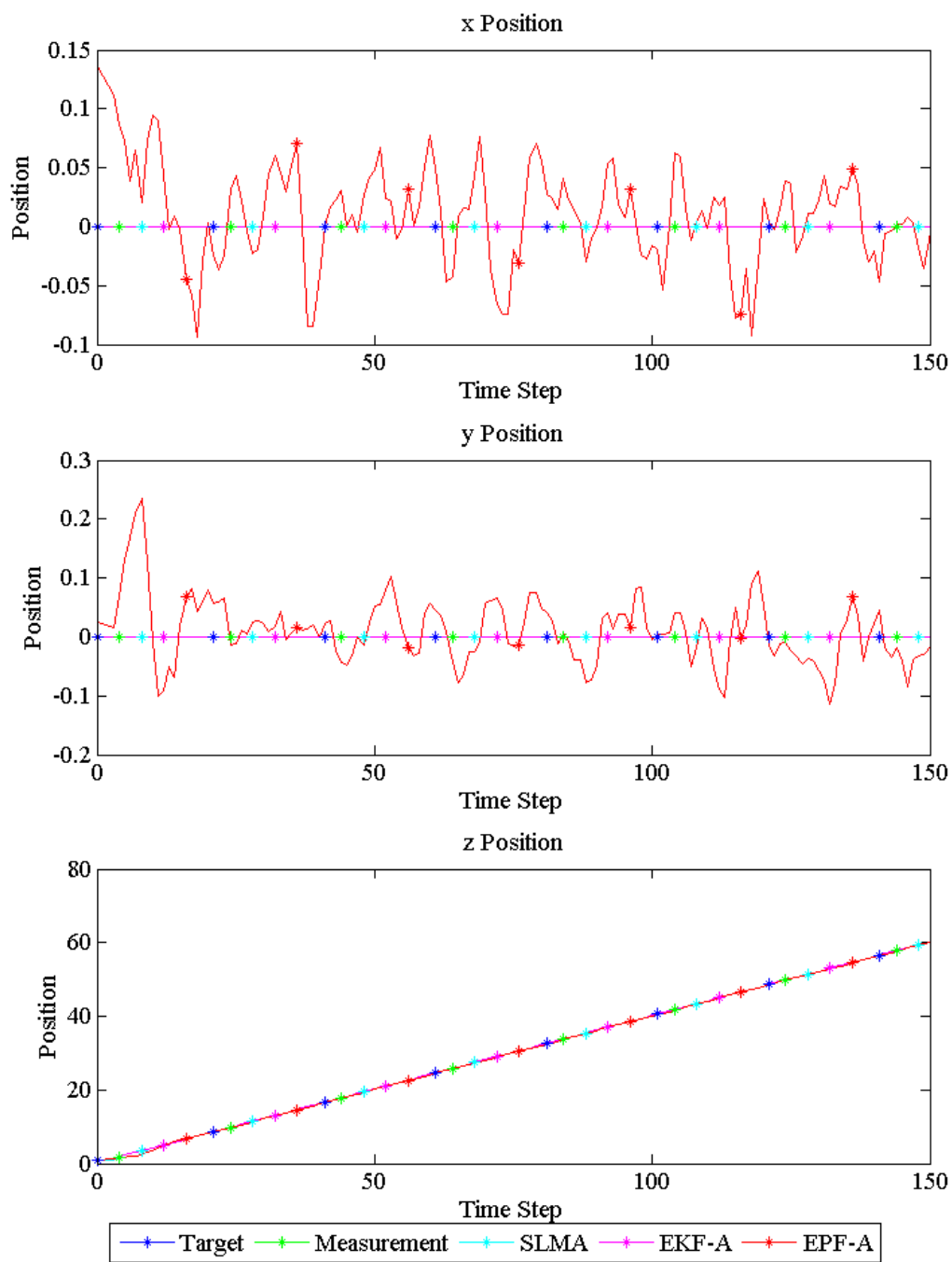


Figure 5.7: Single Simulation for z-Axis Movement without Measurement Noise

Figure 5.3 and 5.4 demonstrate the earlier concerns of the MAE discussed in Section 5.2.1.1, that is that the MAE may obscure the filter's actual performance. Additionally, Figure 5.5, 5.6, and 5.7 show the oscillatory nature of EPF-A based on its model, discussed in Section 5.3. EPF-A's model could be optimized for a linear model, though for that the Kalman filter is more suitable. Table 5.5 details both the overall mean TER and MAE values for each variable for the last 50 time units. The last 50 were used since by this point, the EPF-A was tracking the target and no longer searching as it was within approximately the first 25 time units.

Table 5.5: Mean Axial Errors for Last 50 Time Steps without Measurement Noise

Metrics	D	V	θ	ϕ
Mean Error				
SLMA X-Axis	0.0000	0.0000	0.0000	0.0000
EKF-A X-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A X-Axis	0.0845	0.1970	3.1749	3.2342
SLMA Y-Axis	0.0000	0.0000	0.0000	84.7059
EKF-A Y-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A Y-Axis	0.0708	0.2322	21.1460	90.6877
SLMA Z-Axis	0.0000	0.0000	0.0000	0.0000
EKF-A Z-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A Z-Axis	0.0837	0.2063	3.2913	3.2316
Threshold Error				
SLMA X-Axis	0.0000	0.0000	0.0000	0.0000
EKF-A X-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A X-Axis	0.0534	0.3775	6.6001	6.4208
SLMA Y-Axis	0.0000	0.0000	0.0000	130.5882
EKF-A Y-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A Y-Axis	0.1319	0.4792	23.1609	158.6222
SLMA Z-Axis	0.0000	0.0000	0.0000	0.0000
EKF-A Z-Axis	0.0000	3.6000	0.0000	0.0000
EPF-A Z-Axis	0.1455	0.4140	6.6470	6.6012

Overall the TER is greater than MAE, indicating that there are not many distant outliers. A common trend was that the TER was roughly twice the MAE. Additionally, since no noise is present and the movement is completely linear (straight line, constant velocity), the SLMA performs better than EPF-A. Furthermore, due to the stochastic nature of the filter as discussed in Section 5.2, EPF-A has a near zero likelihood of having zero error. In order to further verify that the distribution is regular, the TER for several different percentile values was calculated and plotted in Figure 5.8.

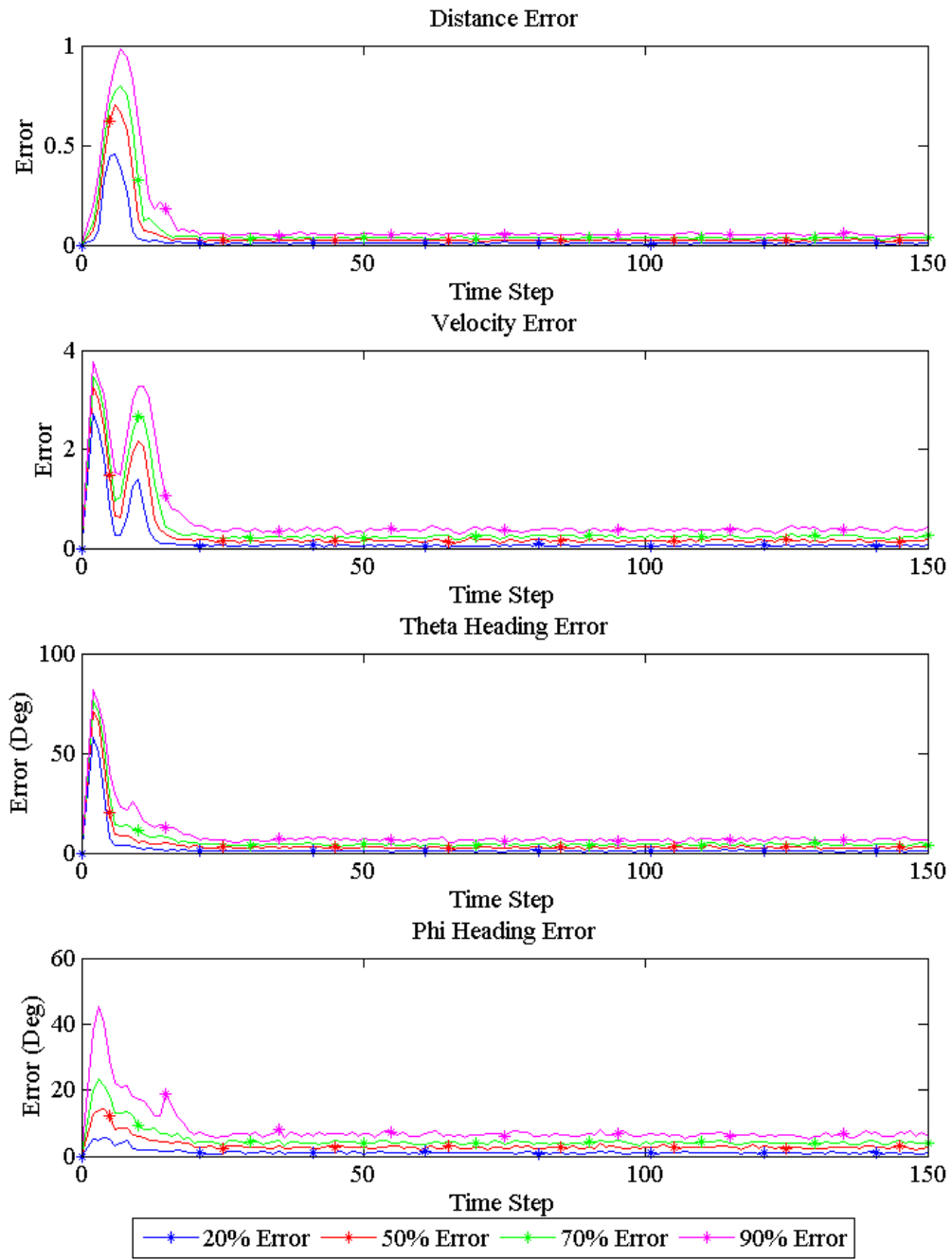


Figure 5.8: Comparison of Threshold Values for X-Axis Movement with No Measurement Noise

As seen in Figure 5.8, the TERs showed regularly increasing error with increasing percentile. Since they did not all tend towards a common nonzero value, this indicated there is little chance of bias being present. Alternatively, bias did exist within the first 25 time steps when the filter was tracking the target. This bias existed due to incorrect initial conditions. Also of note is the increase in θ heading error present in EPF-A for movement along the y-axis. In theory, this error should cause an increase in position error since the direction for speed is not in the direction of the target. For this scenario, the target θ should be equal to 90, which is equal to the target value. However, the cosine of 18.7662 and 16.7740 are 0.9468 and 0.9613 respectively. When the variance for θ , 0.1, is added during the importance sampling step, particles that match the target θ are generated, thus allowing the particle filter to continue tracking the target despite the heading error. The errors in ϕ are due to the presence of singularity about the y-axis. The potential for less accurate particles to be chosen increases, since one of the measurements and hence selection criteria, ϕ , is meaningless and cannot be used as a discriminator. This is not a flaw on the particle filter, but rather a flaw inherent in the parametrization of the variables. In order to avoid singularities, and increase accuracy, an alternate method of parametrization, such as quaternions, should be used. Subsequent scenarios introduced measurement noise as seen in Figure 5.9 and 5.10. Figure 5.9 contains the MAE values with noise for EPF-A and SLMA and Figure 5.10 contains the TER with noise. A single run for movement about each axis is shown in Figure 5.11, 5.12, and 5.13.

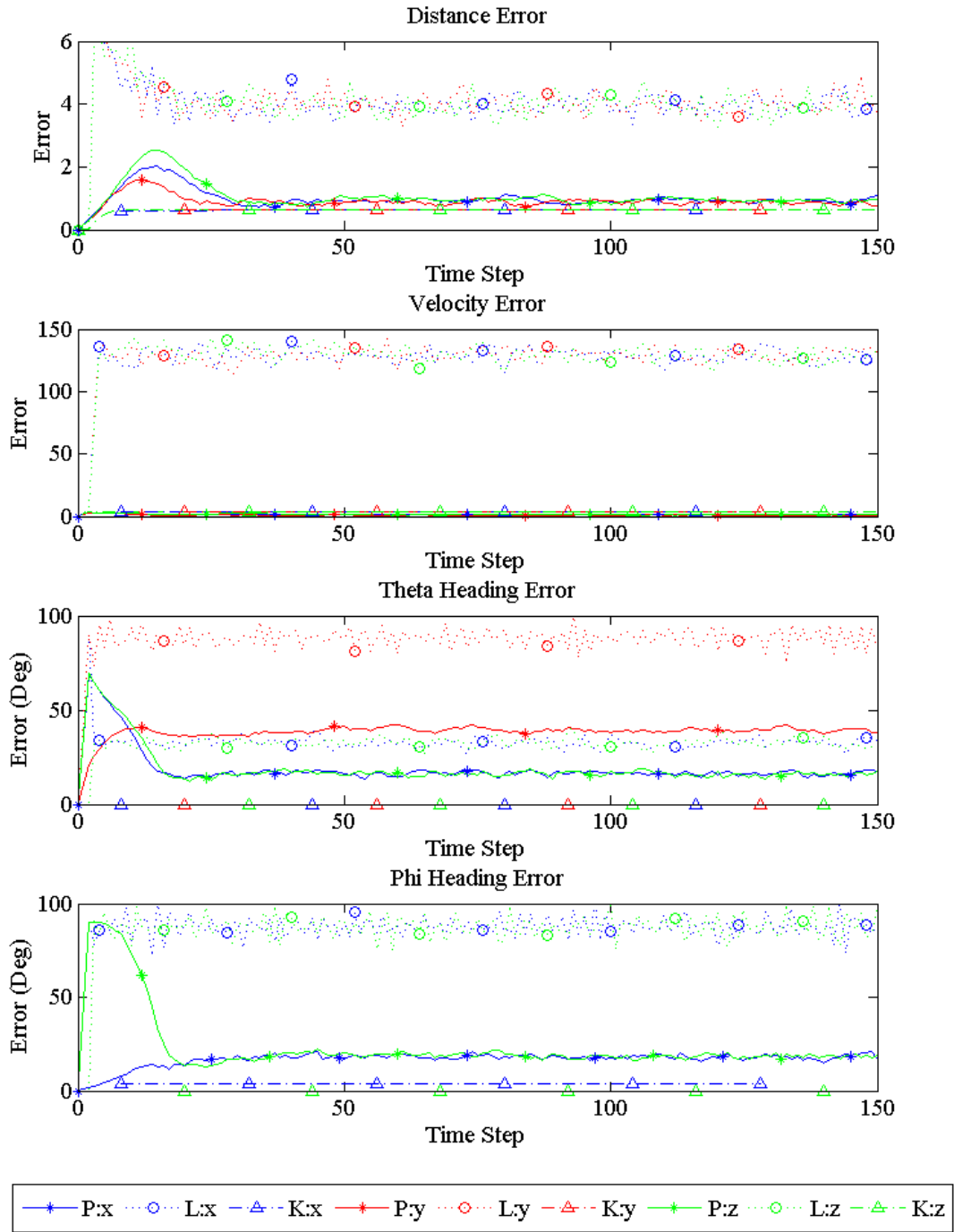


Figure 5.9: Mean Error for Axial Movement with Measurement Noise

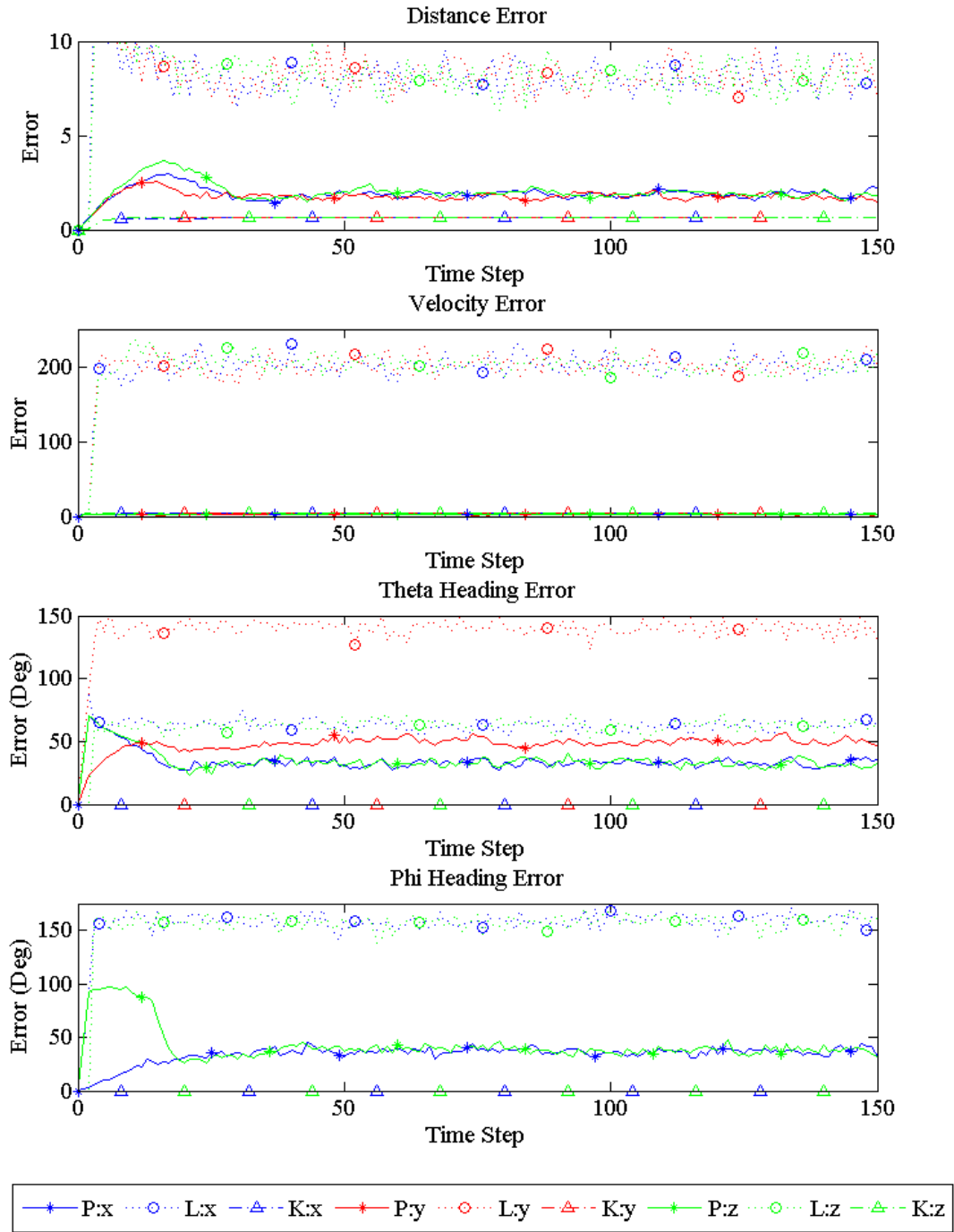


Figure 5.10: Threshold Error for Axial Movement with Measurement Noise

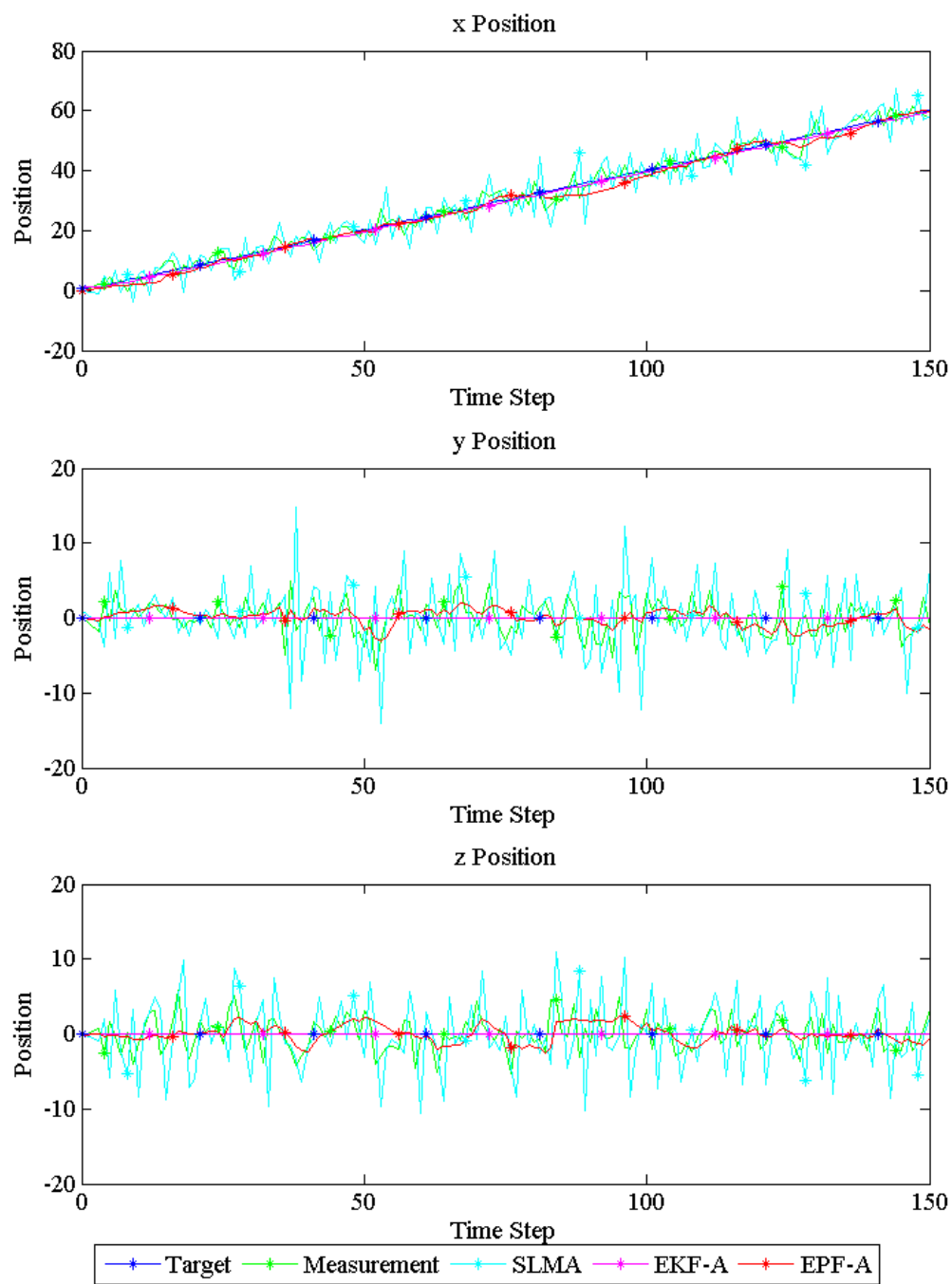


Figure 5.11: Single Simulation for x-Axis Movement with Measurement Noise

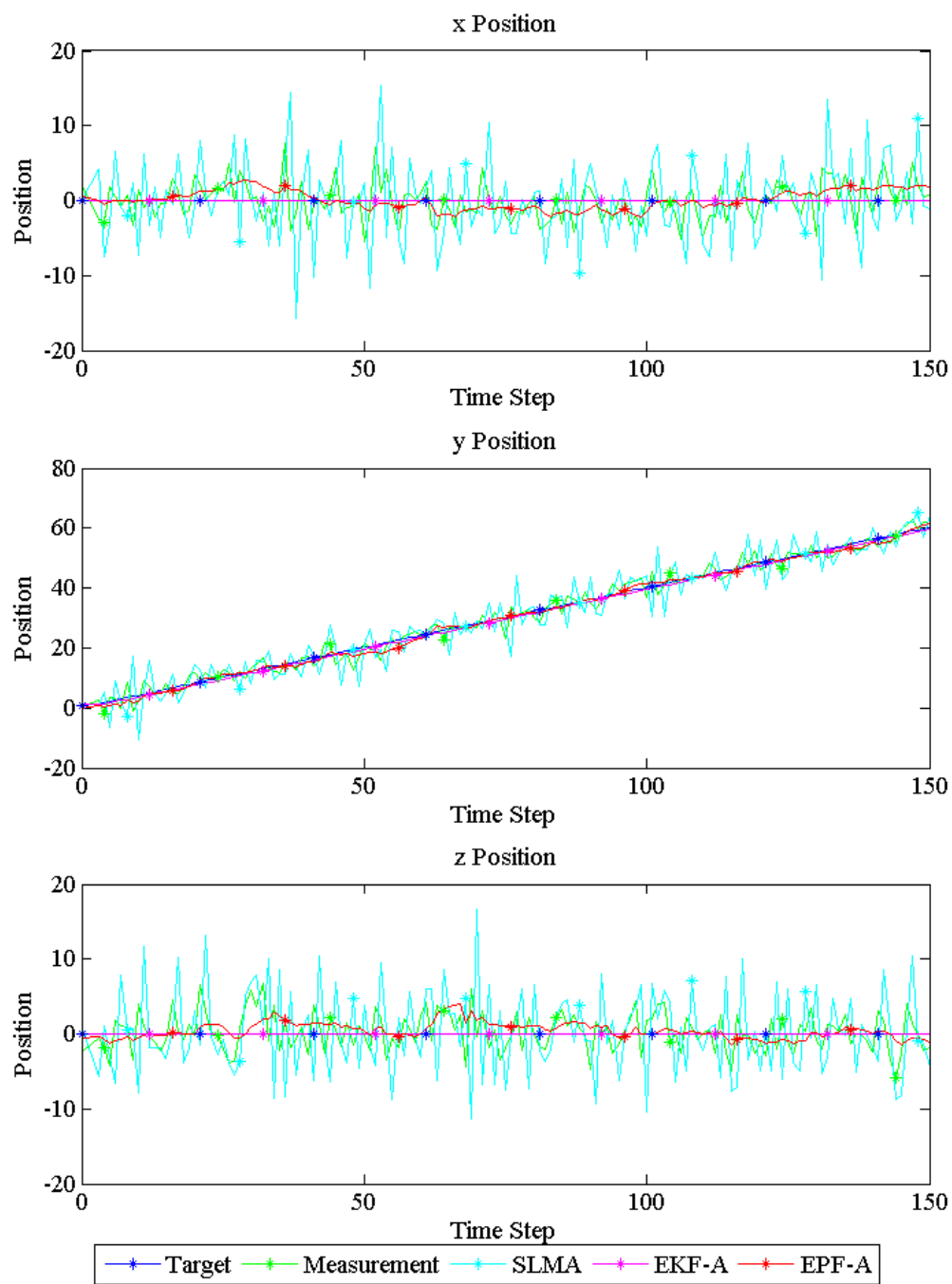


Figure 5.12: Single Simulation for y-Axis Movement with Measurement Noise

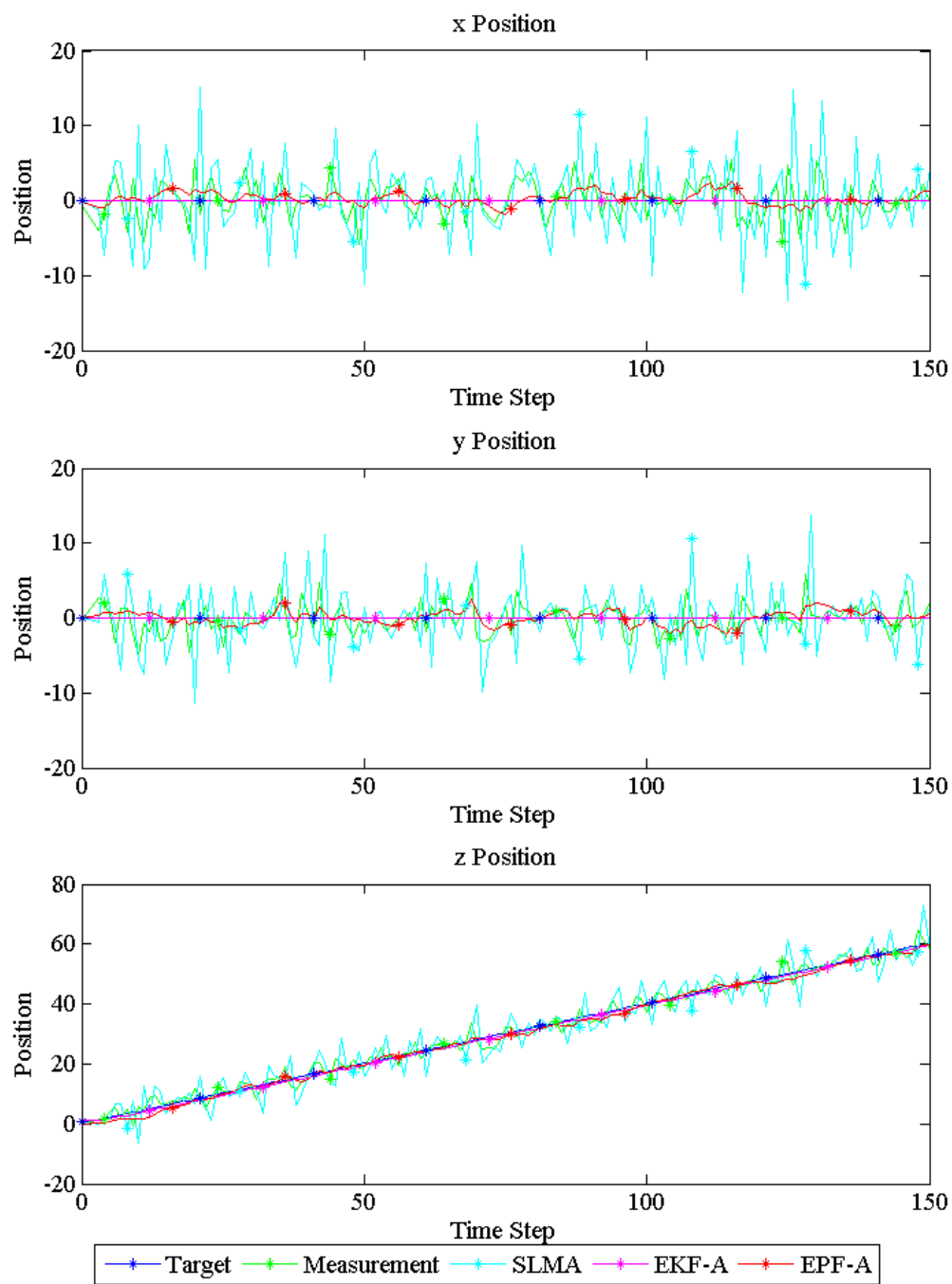


Figure 5.13: Single Simulation for z-Axis Movement with Measurement Noise

With the addition of measurement noise, the SLMA performs significantly worse than the EPF-A. Table 5.6 details both the overall mean TER and MAE values for each variable for the last 50 time units. The last 50 were used since by this point, the EPF-A was tracking the target and no longer searching as it was within approximately the first 25 time units. The EKF-A still exceeds the performance of EPF-A though this is not unexpected given the linear movement of the target. Of note is the noticeable increase in ϕ heading for movement in the y direction. The cause of this is linked to the singularity around the y -axis for θ .

Table 5.6: Mean Axial Errors for Last 50 Time Steps with Measurement Noise

Metrics	D	V	θ	ϕ
Mean Error				
SLMA X-Axis	3.9480	127.9528	32.5460	87.5482
PKF-A X-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A X-Axis	0.9270	1.1662	16.2461	18.0057
SLMA Y-Axis	4.0027	129.8334	88.3252	90.0457
PKF-A Y-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A Y-Axis	0.8610	0.9681	39.2148	89.4936
SLMA Z-Axis	3.9668	128.4708	32.3721	88.2372
PKF-A Z-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A Z-Axis	0.9425	1.1267	15.9382	18.3654
Threshold Error				
SLMA X-Axis	8.0090	199.9310	62.6127	160.7167
PKF-A X-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A X-Axis	1.8840	2.3915	32.8683	36.5628
SLMA Y-Axis	8.1987	204.2919	139.8049	159.6471
PKF-A Y-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A Y-Axis	1.7465	1.9768	50.0713	160.0725
SLMA Z-Axis	8.0351	200.2442	62.6789	158.6492
PKF-A Z-Axis	0.6449	3.6000	0.0000	0.0000
EPF-A Z-Axis	1.8864	2.2948	32.2677	37.7447

5.3.2 Circular Movement.

Six scenarios were conducted: two about each rotation axis where one had perfect measurements and the other had measurement noise. All scenarios ran for 360 iterations, so as to complete one full circular revolution. Table 5.7 lists the initial conditions for each scenario. Measurement noise covariances were the same as those used in the orthogonal axis scenarios, see Table 5.3.

Table 5.7: Circular Rotation Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
Z-Axis Circular Rotation in the XY-Plane												
Target	0	0	20	5	90	0	0	0	0	0	1	0
SLMA	0	0	20	5	90	0	0	0	0	0	0	0
EPF-A	0	0	0	0	0	0	0	0	0	0	0	0
Y-Axis Circular Rotation in the XZ-Plane												
Target	0	0	10	5	90	0	0	0	0	0	0	1
SLMA	0	0	10	5	90	0	0	0	0	0	0	0
EPF-A	0	0	0	0	0	0	0	0	0	0	0	0
X-Axis Circular Rotation in the YZ-Plane												
Target:	0	0	10	5	0	90	0	0	0	1	0	0
SLMA	0	0	10	5	9	90	0	0	0	0	0	0
EPF-A	0	0	0	0	0	0	0	0	0	0	0	0
EPF-A Variance:	1	1	1	1	0.1	0.1	0.5	0.5	0.5	0.1	0.01	0.01

The performance of each circular rotation was evaluated using the performance metrics discussed in Section 5.2.1. Circular rotations were first evaluated without measurement noise. Figure 5.14 contains the MAE values without noise for EPF-A and SLMA and Figure 5.15 contains the TER without noise. A single run for movement about each axis is shown in Figure 5.16, 5.17, and 5.18.

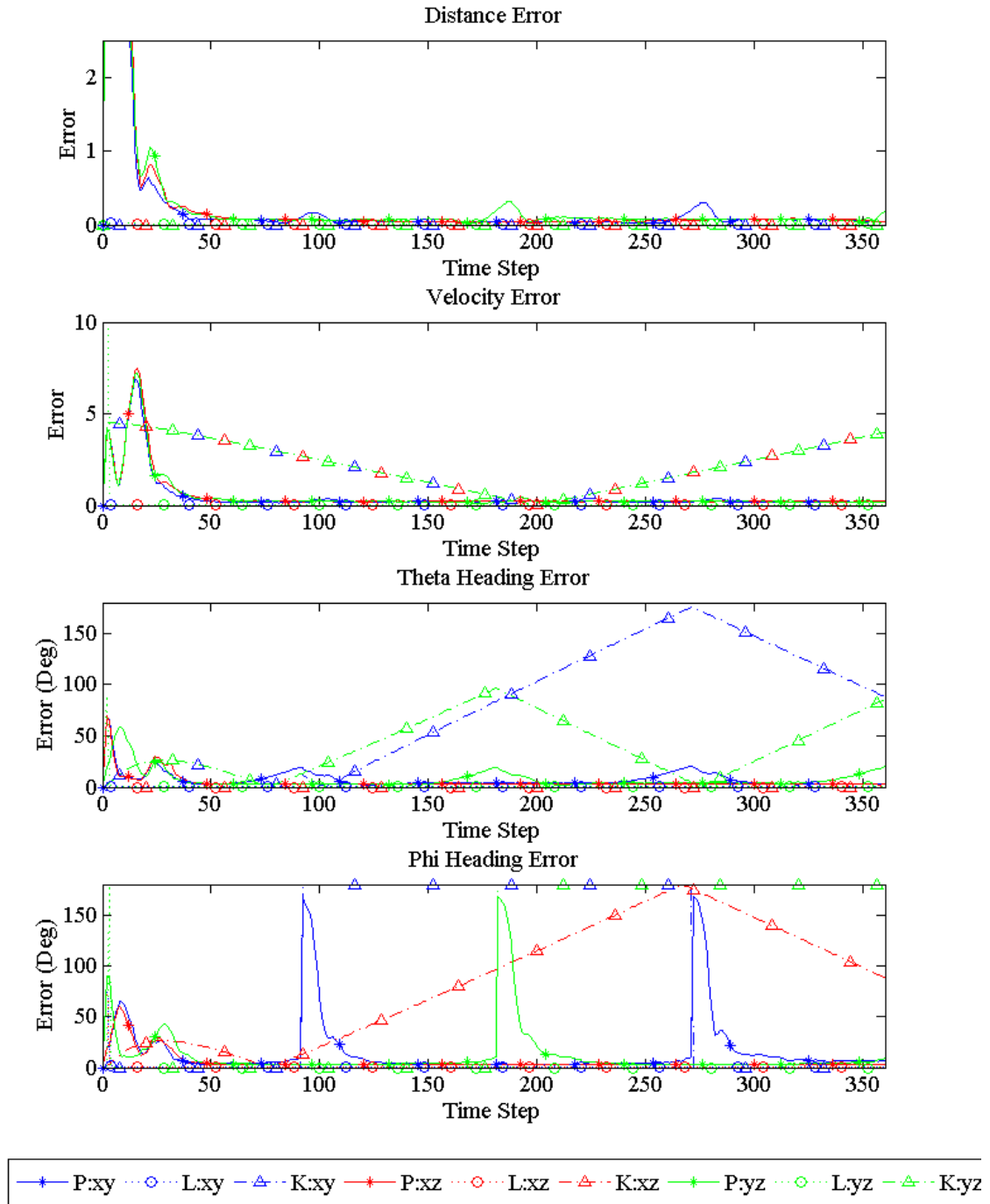


Figure 5.14: Mean Error for Circular Rotation without Measurement Noise

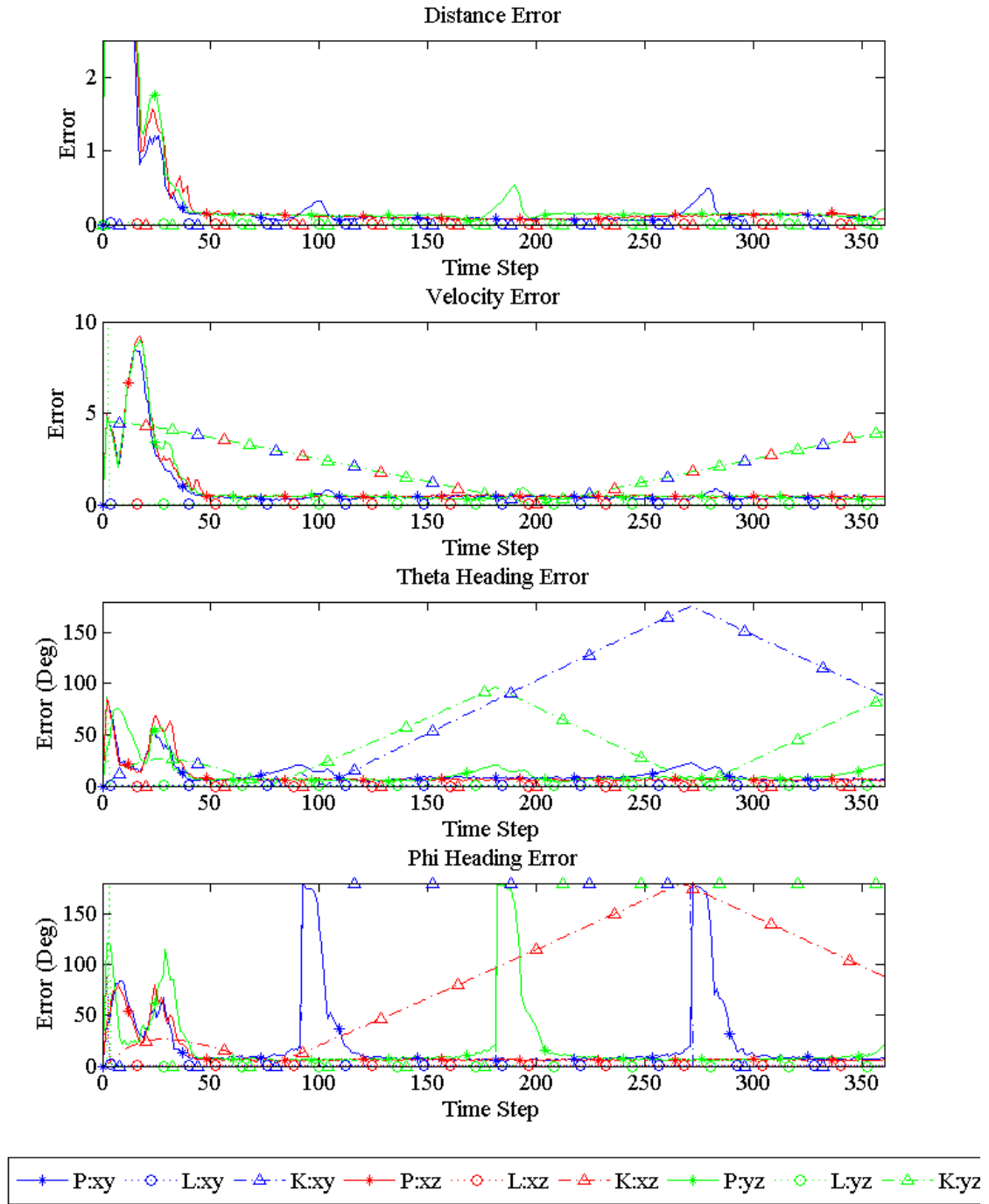


Figure 5.15: Threshold Error for Circular Rotation without Measurement Noise

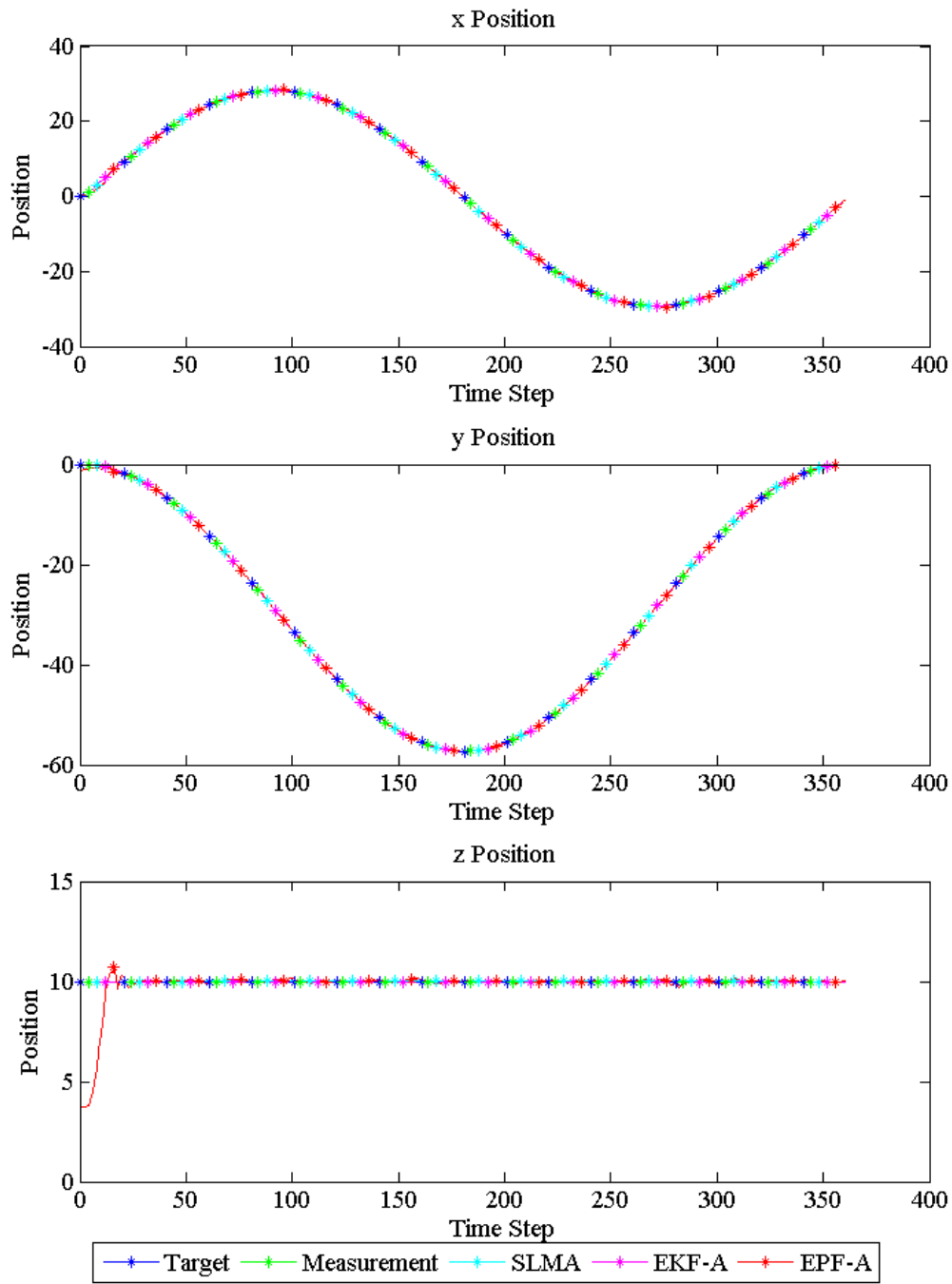


Figure 5.16: Single Simulation for xy-Plane Circular Rotation with Measurement Noise

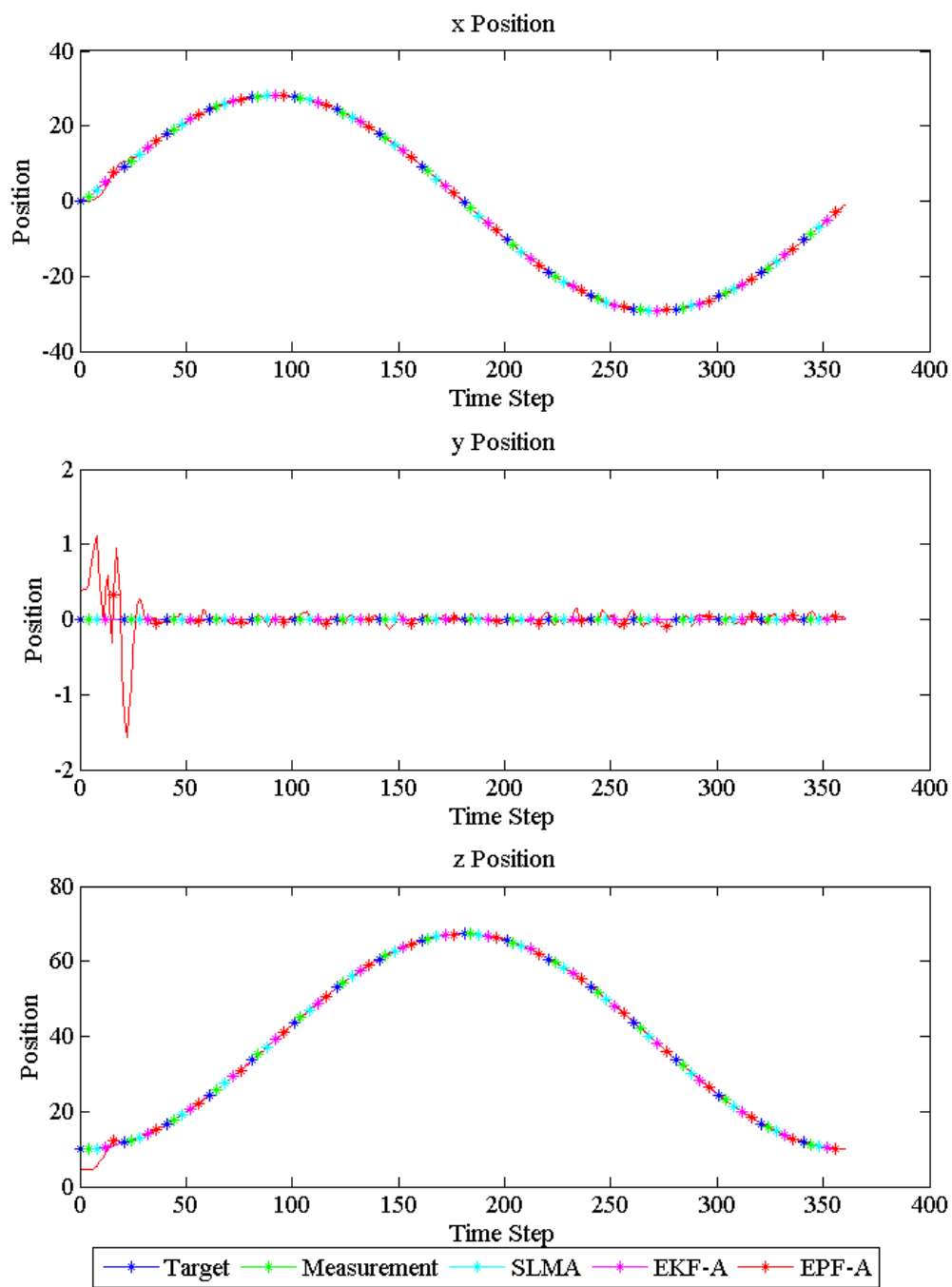


Figure 5.17: Single Simulation for xz-Plane Circular Rotation with Measurement Noise

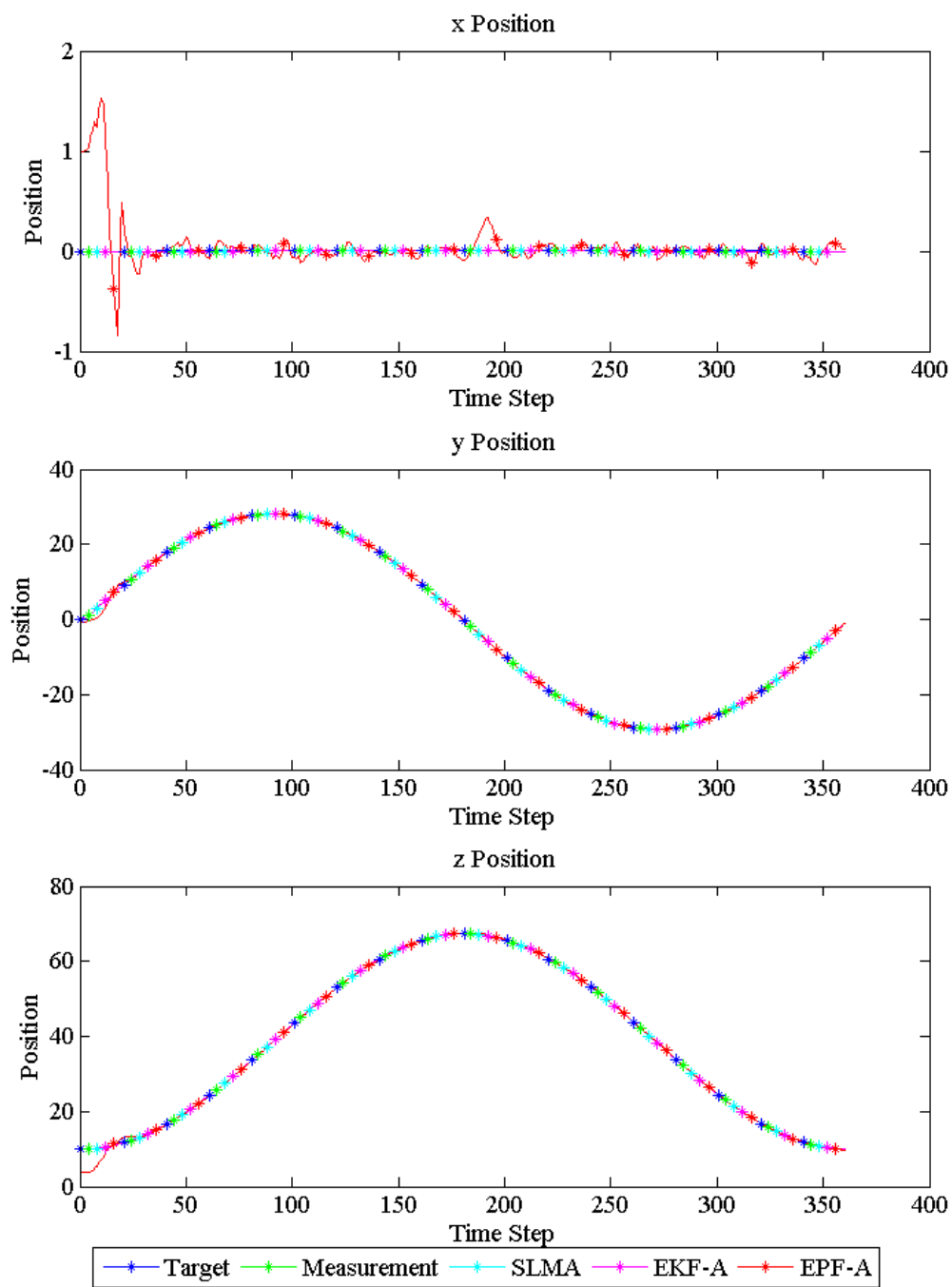


Figure 5.18: Single Simulation for yz-Plane Circular Rotation with Measurement Noise

Although the SLMA performs better than the EPF-A for position estimates without noise, its estimation of the velocity, V , and headings, θ and ϕ , are noticeably worse than the EPF-A. This is due to the sinusoidal, and thus non-linear, motion of the target. Overall, the EPF-A does perform better than the SLMA except for a few noticeable sudden spikes in the heading angles. Similarly to the singularity for the y-axis, these spikes are due to parametrization. The spikes occur when θ or ϕ reach the end of their range and must instantaneously sweep to the other end of its range. Figure 5.19 illustrates the relationship between the target and EPF-A variables for the three circular scenarios without measurement noise. EKF had comparable performance to EPF-A for position estimates, sometimes worse, sometimes better depending on the metric used. However, EKF-A encountered significant errors when estimating the hidden states, velocity and headings. The cause of this error stems from the type of Kalman filter EKF-A is. EKF-A is an optimal gain estimator Kalman filter; this means that it will optimize its Kalman gains for states with corresponding measurements. Since velocity and headings do not have measurements, EKF-A will not return accurate predictions. Rather, it will continue to make predictions based on the initial conditions and control law. For instances in which the hidden states do not change, such as the axial tests (Section 5.3.1), the predictions will be accurate. However, as soon as the hidden states change from their initial conditions, EKF-A's predictions will begin to become inaccurate as demonstrated in this section.

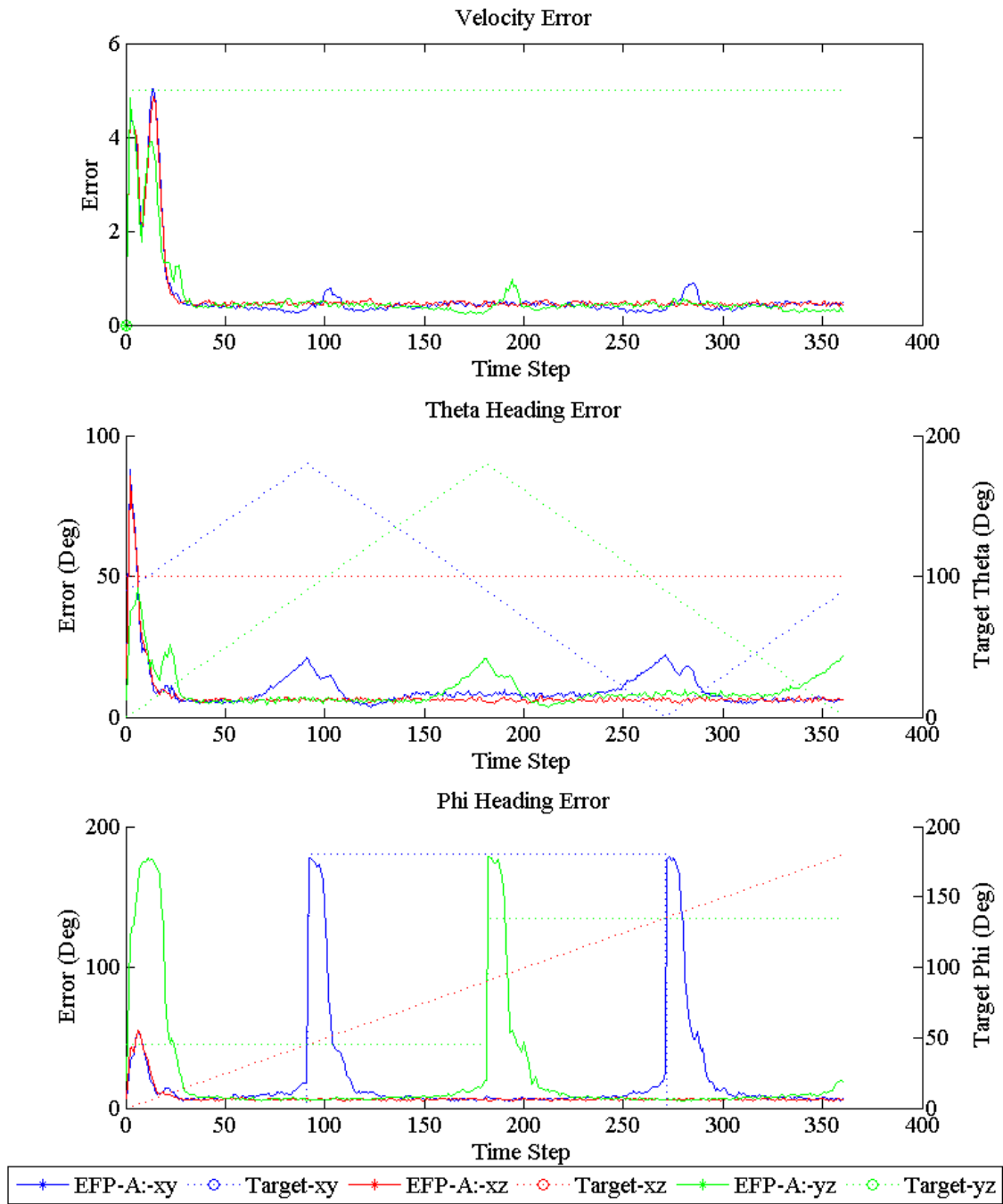


Figure 5.19: Target and EPF-A Variable Relationships

In order to generate mean values of the TER and MAE, variable time ranges are used depending on the scenario to avoid capturing error due to the parametrization and not explicitly the filter itself. The ranges are used for each scenario are as detailed in Table 5.8. Table 5.9 details the mean values of TER and MAE for each variable using variable time ranges of 50 time units.

Table 5.8: Circular Motion Time Unit Ranges for Mean Error

Scenario	Time Range
XY-Circle	160-210
XZ-Circle	160-210
YZ-Circle	260-310

Table 5.9: Mean Circular Errors for Variable Time Steps without Measurement Noise

Metrics	D	V	θ	ϕ
Mean Error				
SLMA XY-Circle	0.0043	0.0006	0.5000	0.0000
EKF-A XY-Circle	0.0000	0.4264	87.7572	180.0000
EPF-A XY-Circle	0.0388	0.2188	4.0210	3.0164
SLMA XZ-Circle	0.0050	0.0006	0.0000	0.5000
EKF-A XZ-Circle	0.0000	0.4264	0.0000	100.2428
EPF-A XZ-Circle	0.0353	0.2289	3.0135	3.0276
SLMA YZ-Circle	0.0027	0.0007	0.4991	0.0000
EKF-A YZ-Circle	0.0000	2.1176	14.8507	180.0000
EPF-A YZ-Circle	0.0749	0.2180	3.9874	3.0637
Threshold Error				
SLMA XY-Circle	0.0043	0.0006	0.5000	0.0000
EKF-A XY-Circle	0.0000	0.4264	87.7572	180.0000
EPF-A XY-Circle	0.0792	0.4439	7.9134	6.0172
SLMA XZ-Circle	0.0050	0.0006	0.0000	0.5000
EKF-A XZ-Circle	0.0000	0.4264	0.0000	100.2428
EPF-A XZ-Circle	0.0717	0.4700	6.0365	6.0905
SLMA YZ-Circle	0.0027	0.0007	0.4991	0.0000
EKF-A YZ-Circle	0.0000	2.1176	0.0000	160.9632
EPF-A YZ-Circle	0.1353	0.4381	7.7922	6.1927

Similarly to the orthogonal movements, the addition of noise continues to degrade the performance of the SLMA while minimally affecting the EKF-A (for observable states) and EPF-A in comparison. Figure 5.20 contains the MAE values with noise for EPF-A and SLMA and Figure 5.21 contains the TER with noise. A single run for movement about each axis is shown in Figure 5.22, 5.23, and 5.24.

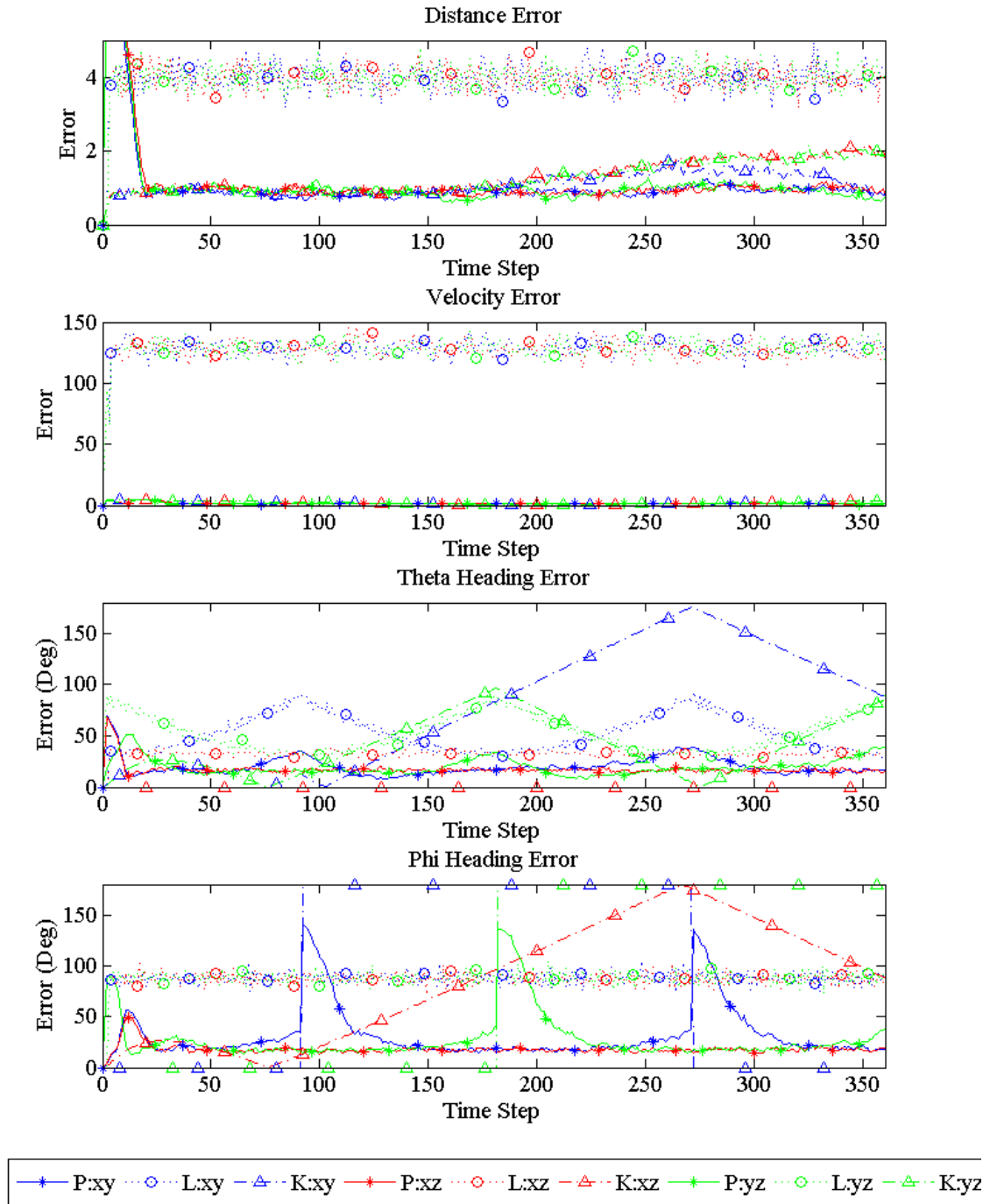


Figure 5.20: Mean Error for Circular Rotation with Measurement Noise

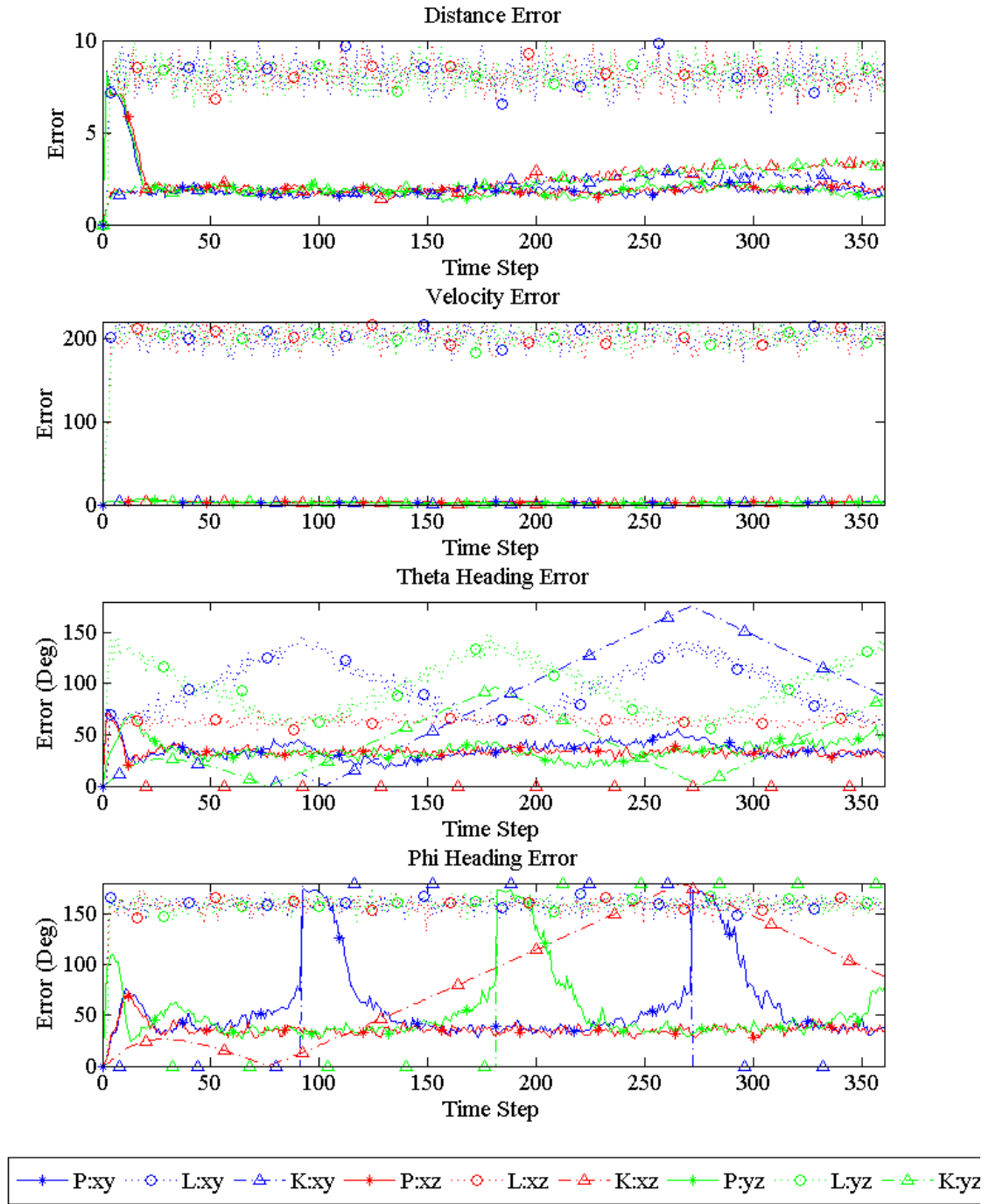


Figure 5.21: Threshold Error for Circular Rotation with Measurement Noise

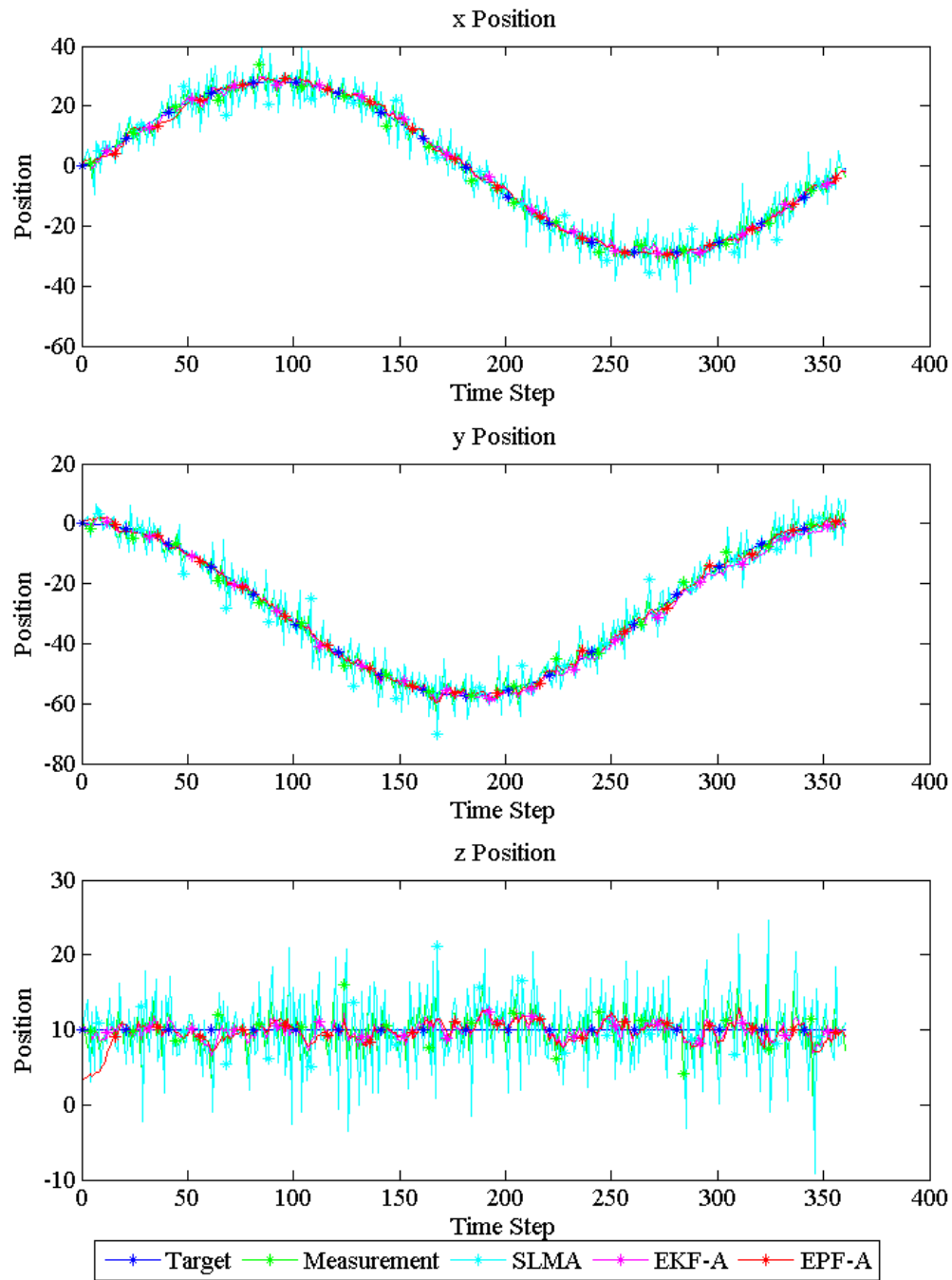


Figure 5.22: Single Simulation for xy-Plane Circular Rotation with Measurement Noise

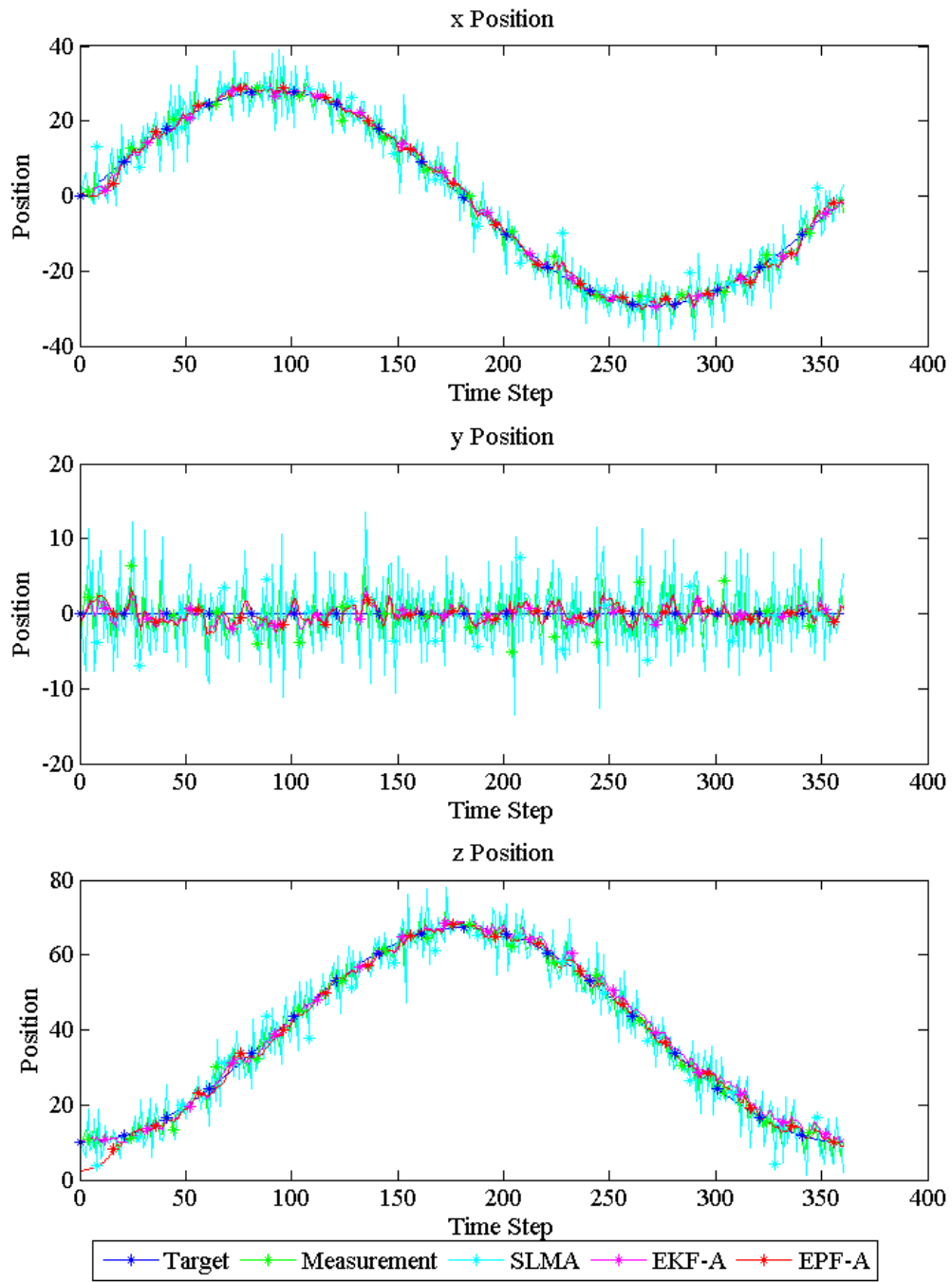


Figure 5.23: Single Simulation for xz-Plane Circular Rotation with Measurement Noise

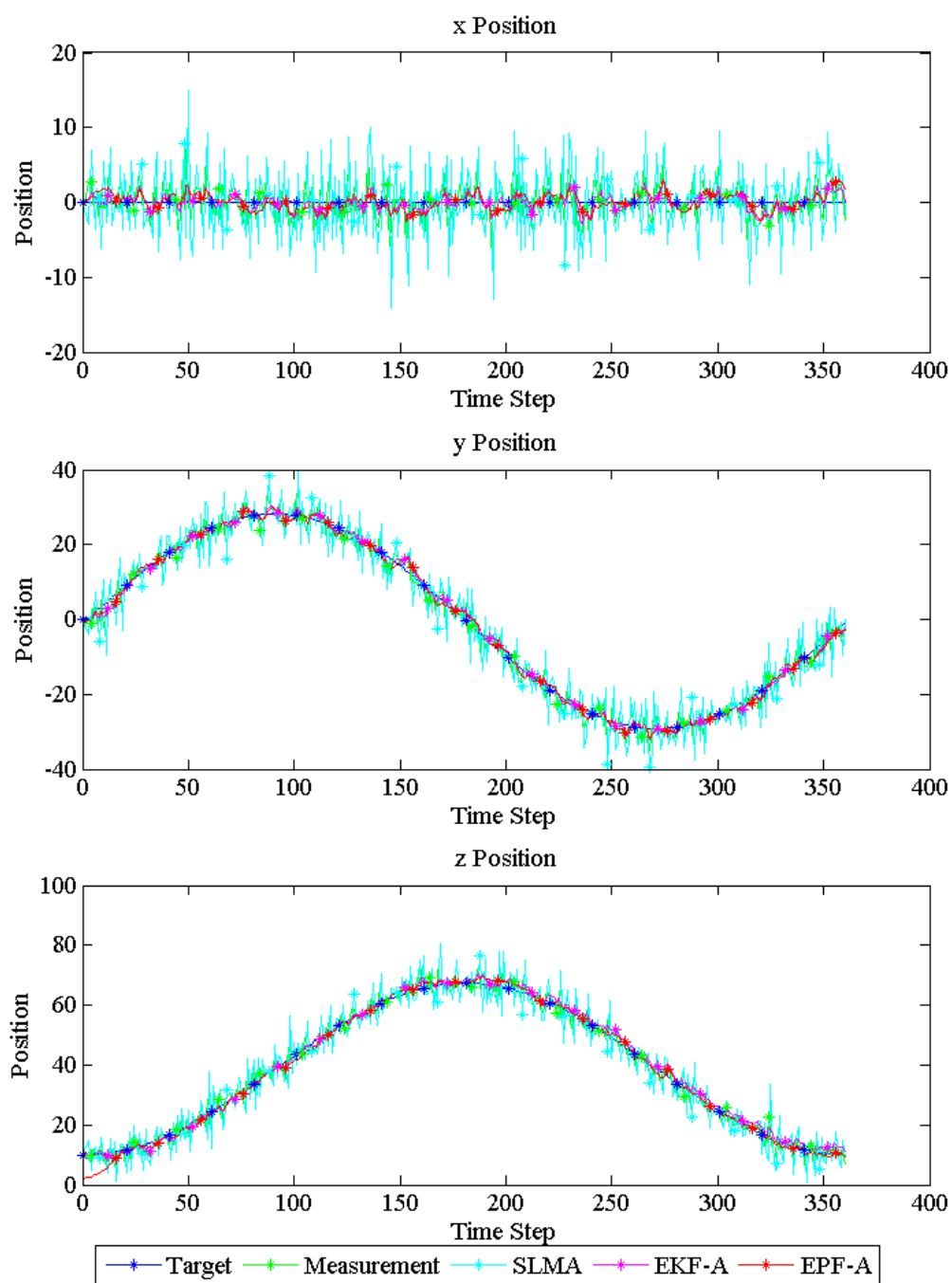


Figure 5.24: Single Simulation for yz-Plane Circular Rotation with Measurement Noise

The addition of noise exacerbates the inaccuracies due to the parametrization. Additionally, the 90 percentile is nearly consistently twice the TER, showing that the shape of the distribution in error changed little between the tests. Table 5.10 details both the overall mean TER and MAE values for each variable using variable time ranges of 50 time units. As with the simulations without noise, variable time ranges are used depending on the scenario to avoid capturing error due to the parametrization and not explicitly the filter itself. The ranges used for each scenario are detailed in Table 5.8.

Table 5.10: Mean Circular Errors for Variable Time Steps with Measurement Noise

Metrics	D	V	θ	ϕ
SLMA XY-Circle	3.9840	128.6077	34.9708	87.1440
EKF-A XY-Circle	1.0912	0.4264	87.7572	180.0000
EPF-A XY-Circle	0.9117	1.4485	17.7539	17.5333
SLMA XZ-Circle	3.9557	128.4116	33.3388	87.4081
EKF-A XZ-Circle	1.1349	0.4264	0.0000	100.2428
EPF-A XZ-Circle	0.8526	1.4320	16.6593	17.3625
SLMA YZ-Circle	4.0371	129.6083	35.9210	87.7064
EKF-A YZ-Circle	1.7615	2.1176	14.8507	180.0000
EPF-A YZ-Circle	1.0621	1.3801	18.2654	17.7514
Threshold Error				
SLMA XY-Circle	8.1087	200.1549	67.8433	158.9436
EKF-A XY-Circle	2.1392	0.4264	87.7572	180.0000
EPF-A XY-Circle	1.8668	2.9561	35.8416	36.4283
SLMA XZ-Circle	8.0401	201.2325	64.2590	159.4563
EKF-A XZ-Circle	2.2251	0.4264	0.0000	100.2428
EPF-A XZ-Circle	1.7261	2.9432	33.6400	35.1490
SLMA YZ-Circle	8.2161	203.0007	70.5211	159.1183
EKF-A YZ-Circle	3.0645	2.1176	0.0000	160.9632
EPF-A YZ-Circle	2.0837	2.8536	36.7118	36.2109

5.3.3 Unconstrained Motion.

Two scenarios were conducted, one without measurement noise and one with, using a set of initial conditions including constant acceleration, detailed in Table 5.11. Table 5.3 details the noise measurement covariances used. The states were unconstrained, meaning the target was not restricted in its movement to a special case, as was done in Section 5.3.1 and 5.3.2.

Table 5.11: Unconstrained Motion Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
Target	5	5	5	14	45	45	0	0	0	0.1	5	5
SLMA	1	0	0	4	90	0	0	0	0	0	0	0
EPF-A	0	0	0	0	0	0	0	0	0	0	0	0
EPF-A Variance:	1	1	1	1	0.1	0.1	0.5	0.5	0.5	0.1	0.01	0.01

The angular acceleration variances were not matched in order to reduce the variety of the heading angles during the re-sampling step of the particle filter. In order to better visualize the motion and results, Figure 5.25 contrasts EPF-A's and SLMA's performance against the target's actual location and the noisy, measured positions.

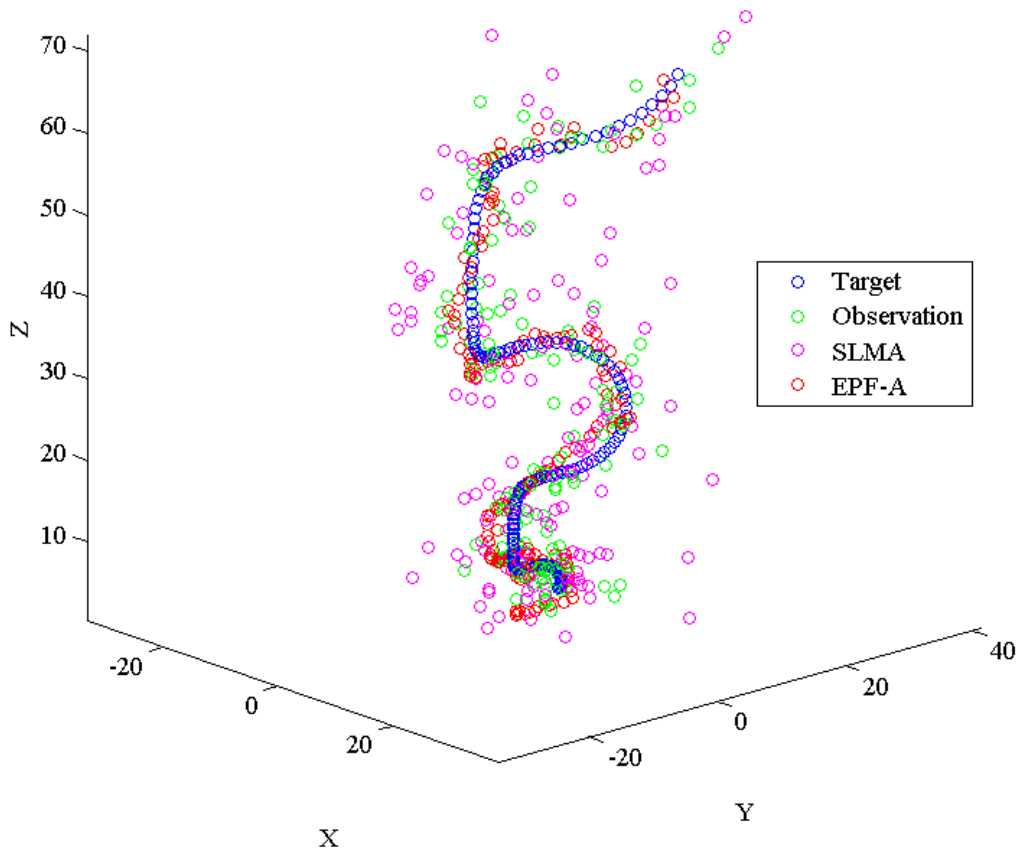


Figure 5.25: Single Simulation for Unconstrained States, in Global Frame, with Measurement Noise

EPF-A typically performed better than SLMA, as expected. The exceptions are when either no measurement noise was present, or for brief periods for the headings θ and ϕ . This is likely due the fact that EPF-A encountered difficulties tracking due to the previously mentioned parametrization flaws. Additionally, for the ϕ heading error, the

SLMA routinely provided a severely erroneous heading, as seen in the TER. Despite the threshold error remaining near 180, the ϕ heading does not have a constant bias since its MAE remained near 90. This shows that the distribution was near uniform. TER and MAE for both the EPF-A and SLMA are seen in Figure 5.27 and 5.26 respectively. Table 5.13 details the overall TER and MAE errors for the EPF-A and the SLMA for the last 50 time units.

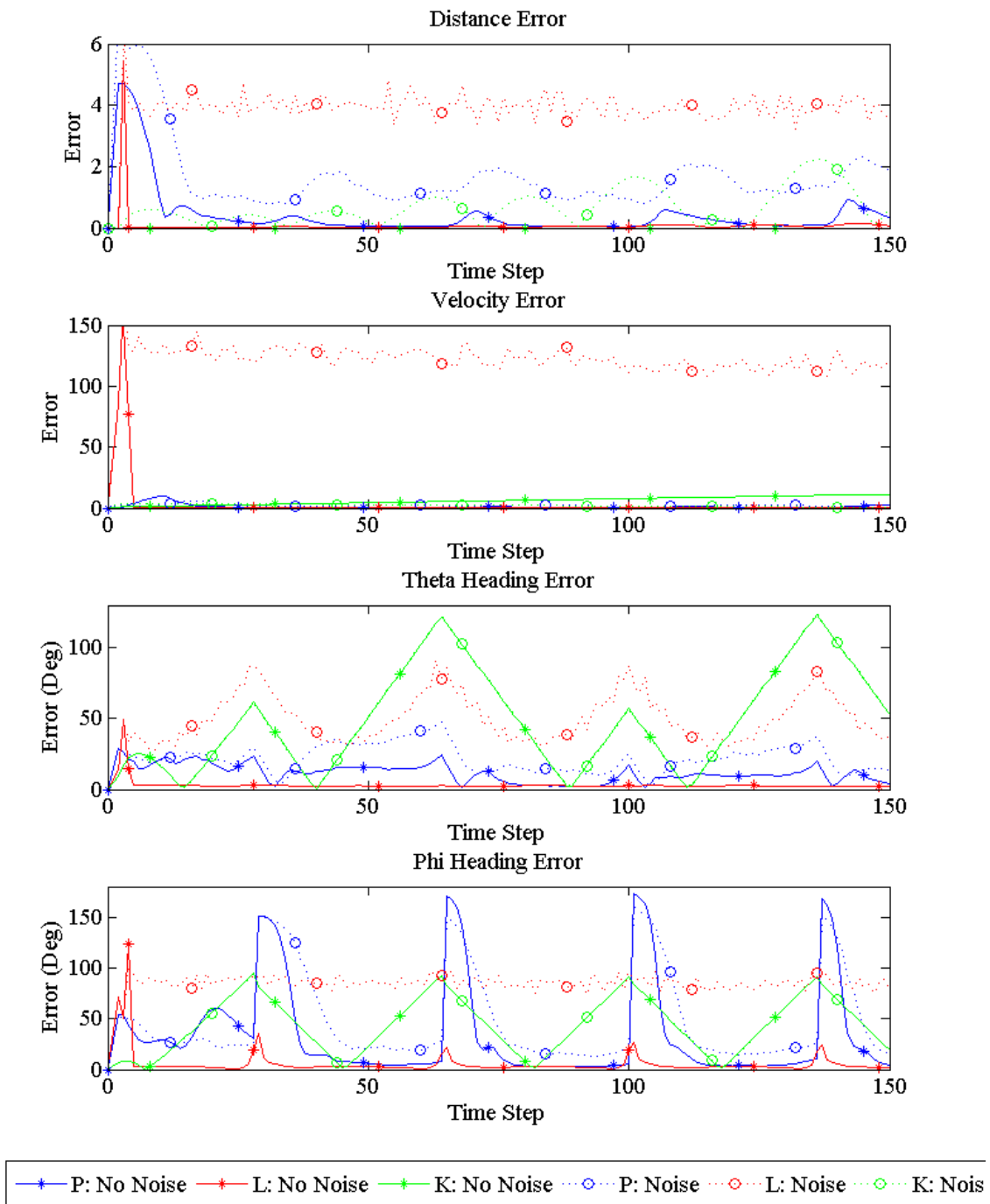


Figure 5.26: Mean Error for Unconstrained Motion

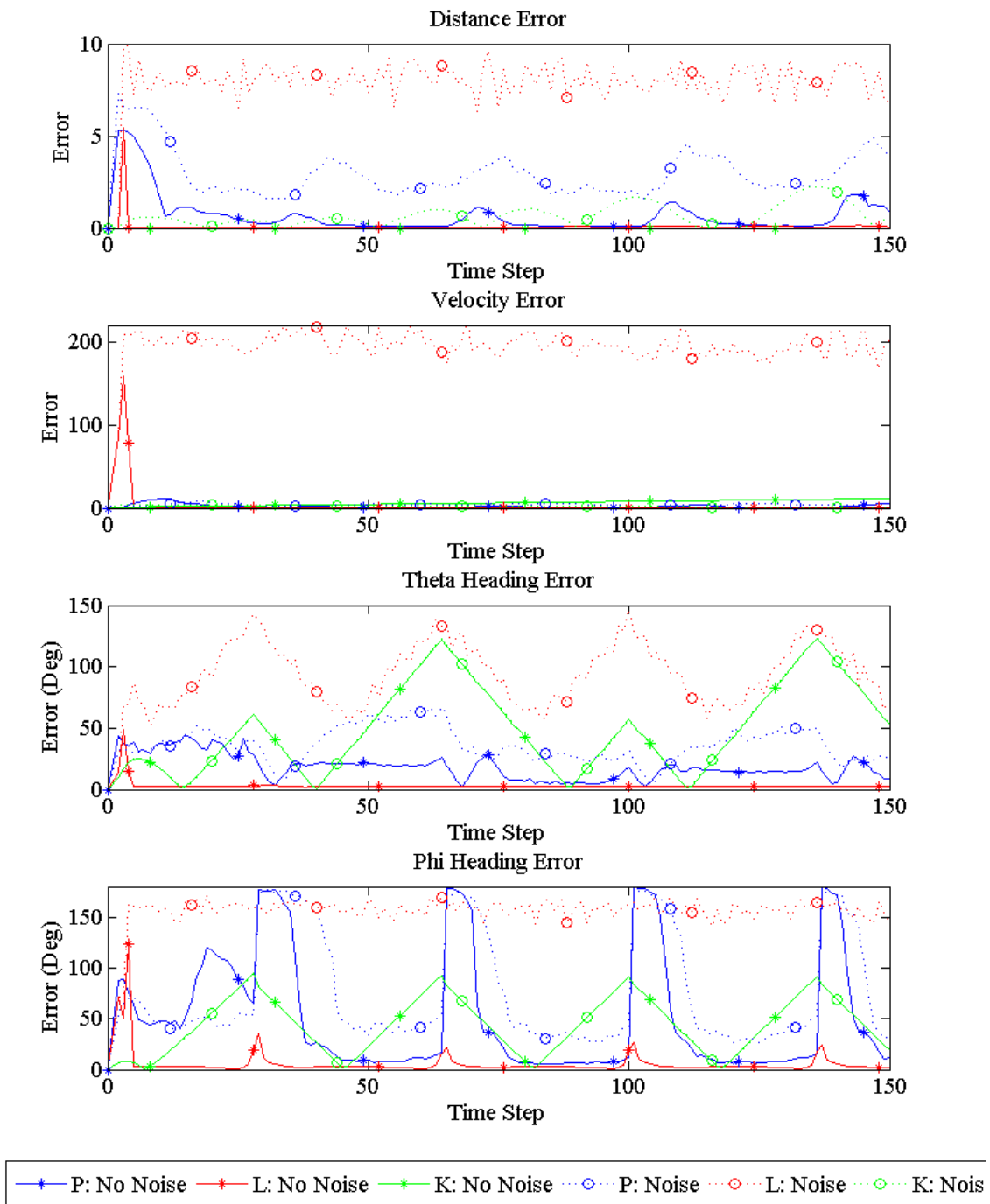


Figure 5.27: Threshold Error for Unconstrained Motion

Table 5.12: Mean Unconstrained Motion Errors for Last 50 Time Steps

Metrics	D	V	θ	ϕ
Mean Error				
SLMA No Noise	0.0778	0.0729	2.5226	4.5982
EKF-A No Noise	0.0000	9.5998	61.4653	47.6136
EPF-A No Noise	0.2877	0.9549	9.4888	37.2105
SLMA With Noise	3.8808	115.9356	51.1688	84.7623
EKF-A With Noise	1.0146	1.2061	61.7888	47.7020
EPF-A With Noise	1.5379	2.0230	20.2282	55.0981
Threshold Error				
SLMA No Noise	0.0778	0.0729	2.5226	4.5982
EKF-A No Noise	0.0000	9.5998	61.4653	47.6136
EPF-A No Noise	0.5909	1.8809	14.7539	53.2478
SLMA With Noise	8.0344	191.7217	97.3950	157.5901
EKF-A With Noise	1.0146	1.2061	61.7888	47.7020
EPF-A With Noise	3.1024	3.9957	32.9966	84.7418

A further distinction between EPF-A and EKF-A is observed with the addition of system noise. As discussed in Section 3.4.5.3, once the control law does not match the target accelerations, Kalman filter performance begins to degrade. 100 simulations were performed with a system noise of 0.1 applied to \dot{V} , $\dot{\theta}$, and $\dot{\phi}$. Figure 5.28 shows the mean error for 100 simulation runs. Since each simulation run differs due to the random system noise, only MAE is calculated.

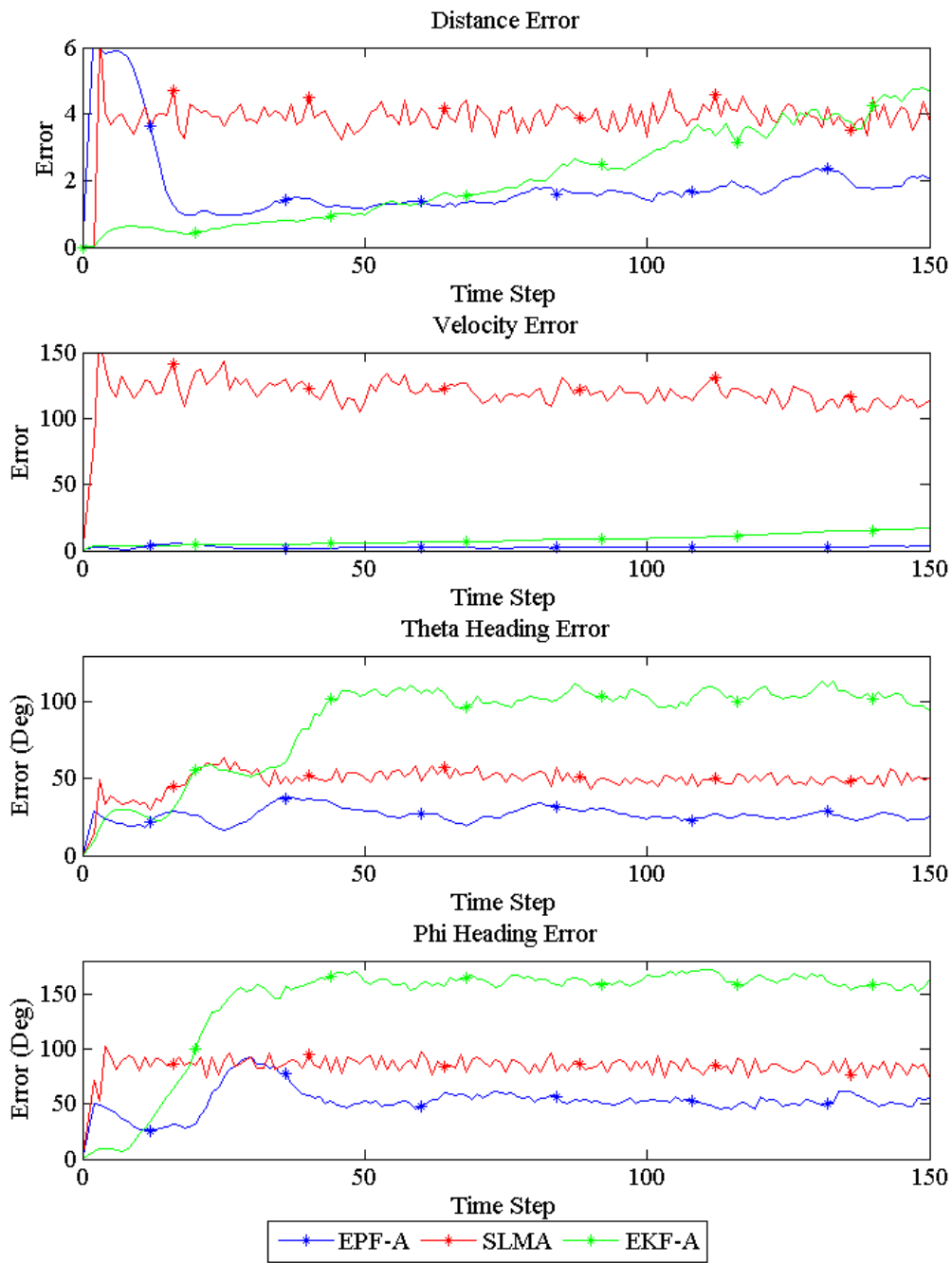


Figure 5.28: MAE for Equal Weight Matrices with System Noise

As expected, EKF-A performs worse than EPF-A due to the inaccuracies of the control law for even slight amounts of system noise. This further emphasizes the particle filter's superiority for predicting movement of non-linear systems when compared to an optimal gain Kalman filter.

Table 5.13: Mean Unconstrained Motion Errors with System Noise for Last 50 Time Steps

Metrics	D	V	θ	ϕ
Mean Error				
SLMA	3.9797	115.5915	49.3769	83.0516
EKF-A	3.7723	12.9709	103.6987	162.5783
EPF-A	1.8712	2.8205	25.2042	51.9195

5.3.4 Evaluated Particle Filter A Summary.

Overall, these scenarios demonstrated the ability of the EPF-A to track a target moving with nonlinear motion, provided only noisy measurements of the target's location. There are a few discrepancies when compared to the Kalman filter, however these errors can be traced to the parameterization effects. Additionally, EPF-A was able to determine hidden target states, velocity magnitude and heading, that could aid the control of a PTZ camera(s) to follow the target more smoothly and accurately.

5.4 Evaluated Particle Filter B

EPF-B simulates the measurements received by a single camera, u_p , v_p , s , \dot{u}_p , and \dot{v}_p , as well as camera pan and tilt angles. Due to the instability of this filter, caused by its frequent propensity to suffer from weight collapse, a single set of initial conditions were used, see Table 5.14. The weight collapse mitigation techniques, discussed in in Section 4.5, were evaluated against these initial conditions. The three weight

mitigation techniques used in Section 5.4 are Non-Compensated Evaluation Particle Filter B (NC), Depth-Compensated Evaluation Particle Filter B (DC), and Jacobian-Compensated Evaluation Particle Filter B (JC). These techniques were tested both with and without noise. These techniques were also tested using two different weighing matrices in an attempt to increase depth accuracy, as discussed in Section 4.5.3. Additionally, the performance of these techniques was contrasted against the SLMB, as well as EPF-A and SLMA. The variables used by EPF-A and SLMA are the same as those used in Section 5.3. Since the same target model is used for both EPF-A and EPF-B, the measurements used by EPF-A and SLMA are extracted before the target model generates the measurements used by EPF-B. The two scenarios tested were:

1. Equally Weighted Weighing Matrix (EW)
2. Unequally Weighted Weighing Matrix (UW)

The values for these matrices are detailed in Equation 5.39.

$$\begin{array}{ccc}
 \left[\begin{array}{ccccc} \Omega(u_p) & 0 & 0 & 0 & 0 \\ 0 & \Omega(v_p) & 0 & 0 & 0 \\ 0 & 0 & \Omega(s) & 0 & 0 \\ 0 & 0 & 0 & \Omega(\dot{u}_p) & 0 \\ 0 & 0 & 0 & 0 & \Omega(\dot{v}_p) \end{array} \right] & \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right] & \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{array} \right] \\
 \text{Weighing Matrix} & \text{Equal Matrix} & \text{Unequal Matrix}
 \end{array} \quad (5.39)$$

The initial conditions of both scenarios are shown in Table 5.14.

Table 5.14: EPF-B Scenario Initial Conditions

States	x	y	z	V	θ	ϕ	Δx	Δy	Δz	ΔV	$\Delta \theta$	$\Delta \phi$
Target	10	10	10	5	10	10	0	0	0	0.2	0.5	0.5
SLMA	10	10	10	5	10	10	0	0	0	0.2	0.5	0.5
EPF-A	10	10	10	4	10	10	0	0	0	0	0	0
SLMB	10	10	10	5	10	10	0	0	0	0.2	0.5	0.5
EPF-B	10	10	10	4	10	10	0	0	0	0	0	0
Filter Variances												
EPF-A	0.1	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1
EPF-B	0.1	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1

The two scenarios also shared several parameters, seen in Table 5.15. The same parameters were used for each filter in-order to allow for direct comparisons. A step size of 0.1 was chosen for the purposes of scaling.

Table 5.15: EPF-B Scenario Parameters

Parameter	EPF-A	EPF-B
Number of Simulations	50	50
Number of Iterations	50	50
Time Step	0.1	0.1
Number of Particles	500	1500

The measurement noises variances used by both scenarios are detailed in Table 5.16.

Table 5.16: Measurement Noise Variations

	No Noise			Noise		
Measurements	u	v	w	u	v	w
Target	0	0	0	5	5	5
EPF-A	0	0	0	5	5	5
EPF-B	0	0	0	5	5	5

5.4.1 Equal Matrix.

As discussed, all three weight collapse variations were tested as well as the SLMB, EPF-A, and SLMB to provide comparison. The TER threshold was 0.9. A single set of initial conditions was used for two scenarios, without and with noise. All results for both scenarios are shown in Table 5.20 to provide direct comparison. The MAE and TER for this specific scenario are shown in Figure 5.29 and 5.30 respectively. The abbreviations used within the legend of each plot are detailed in the list of acronyms, as well as Table 5.17.

Table 5.17: Measurement Noise Variations

Acronym	Meaning
NC	Non-Compensated EPF-B
DC	Depth-Compensated EPF-B
JC	Jacobian-Compensated EPF-B
EW	Equally Weighted Weighing Matrix
UW	Unequally Weighted Weighing Matrix

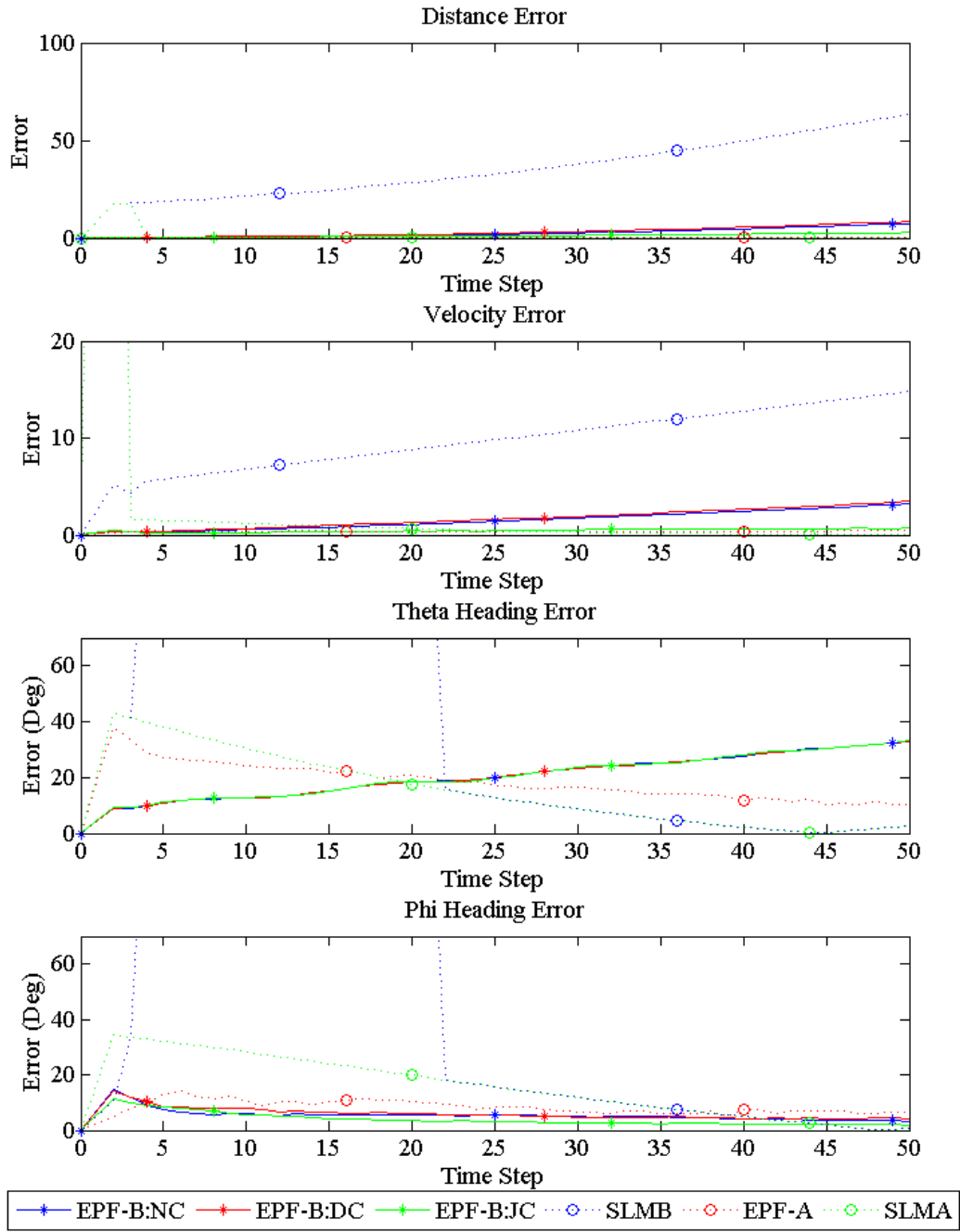


Figure 5.29: MAE for Equal Weight Matrices with No Noise

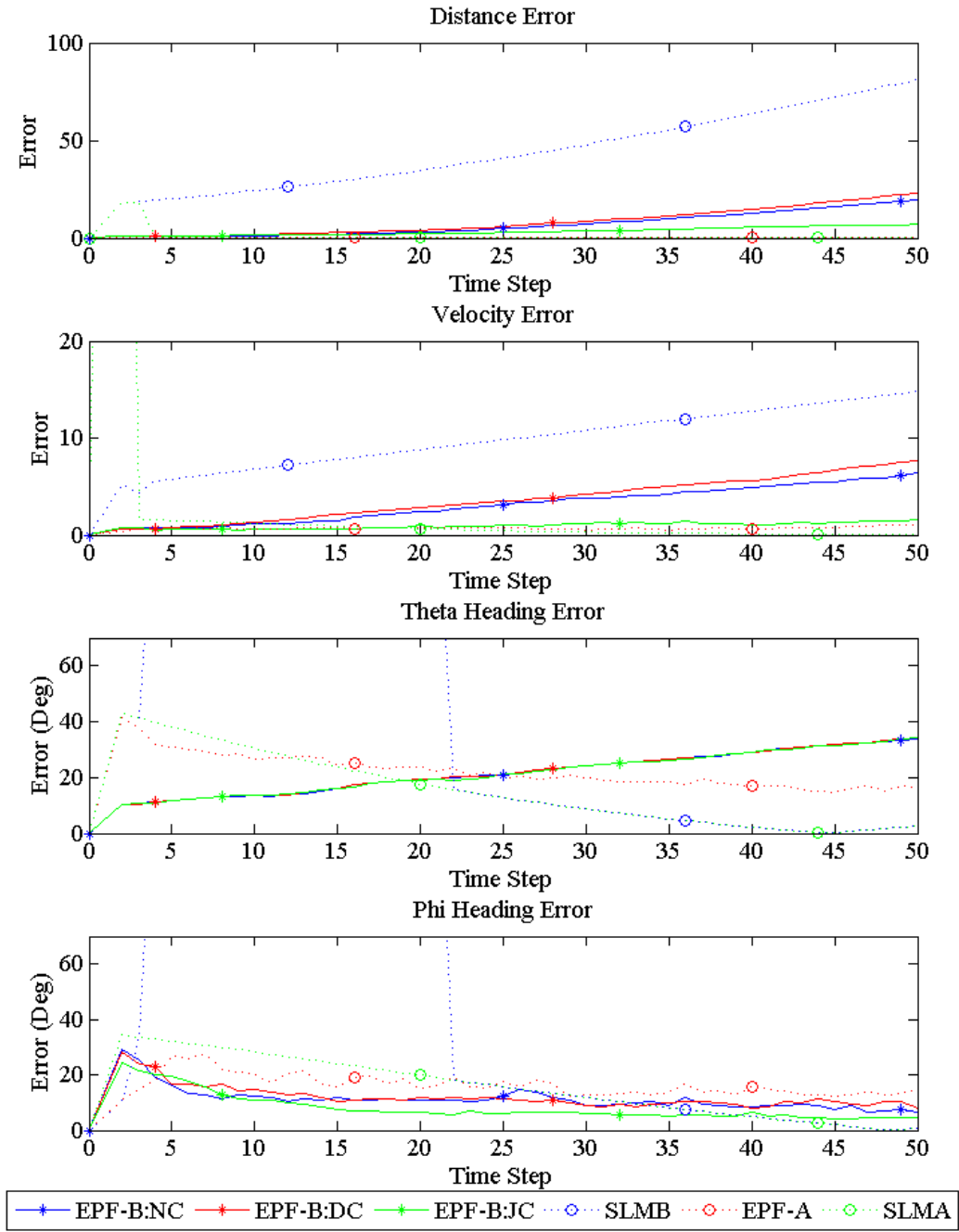


Figure 5.30: TER for Equal Weight Matrices with No Noise

As seen in both Figure 5.29 and 5.30, the performance of the three mitigation methods varied. All three began to diverge in both distance and velocity, due to the EPF-B's inability to continue tracking a target as the velocity continues to increase. The un-mitigated EPF-B and DC had little discernible difference; indeed, for the TER the DC performed worse than the uncompensated model. As for the JC, it's performance was less degraded since it could change its variance to compensate for the increases in velocity. However, this increase in variance appears to have an adverse affect on the heading angles. As expected, the SLMB returned the most erroneous distance and velocity values and continued to worsen as time progressed. However, the SLMB heading errors decreased as the model progressed, but since they decreased in a linear fashion, this decrease may be coincidental since the target is non-linear with changing velocity. Even a linear model should reflect this acceleration via a changing slope. Figure 5.31 and 5.32 show the effects of noise on the various models.

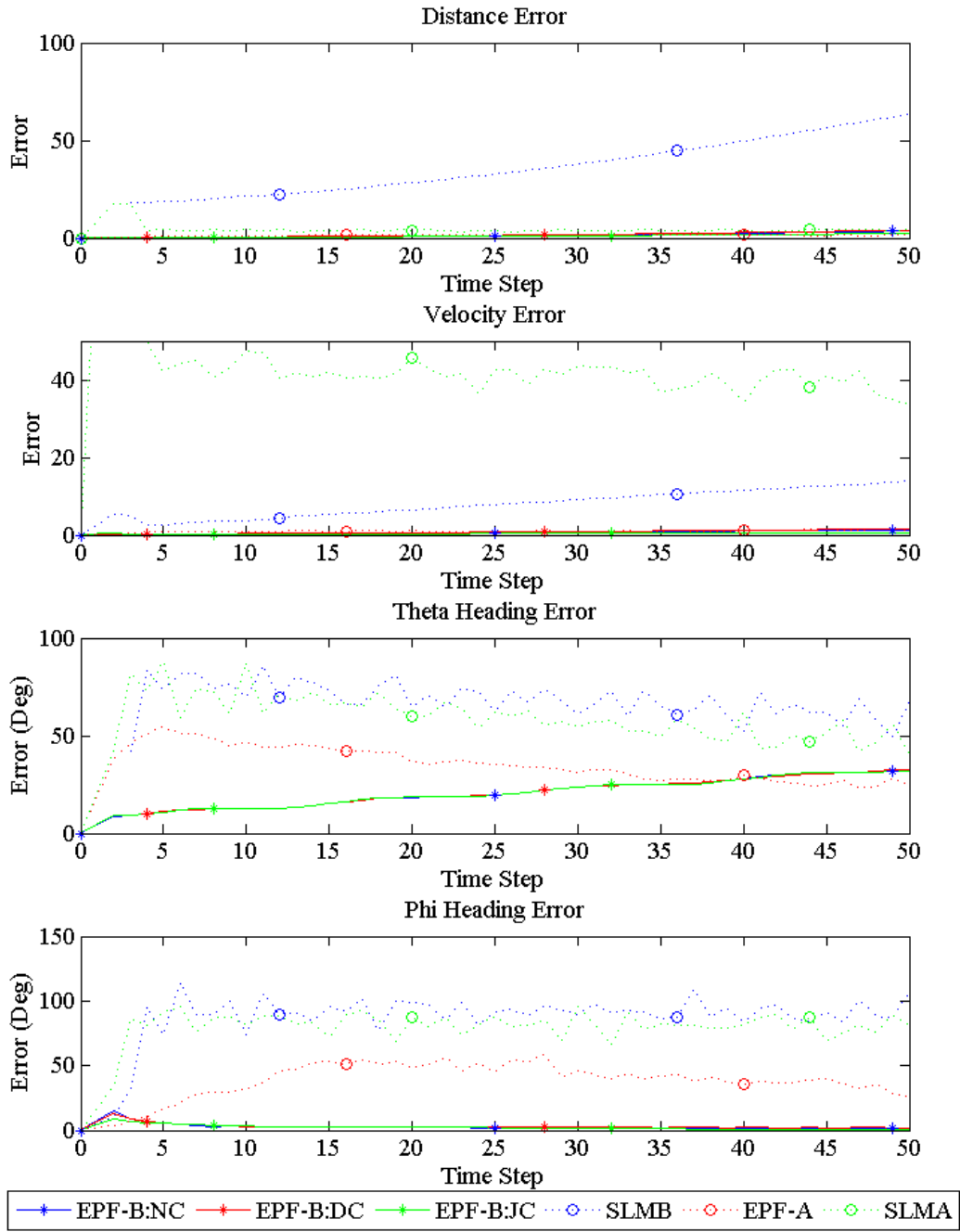


Figure 5.31: MAE for Equal Weight Matrices with Noise

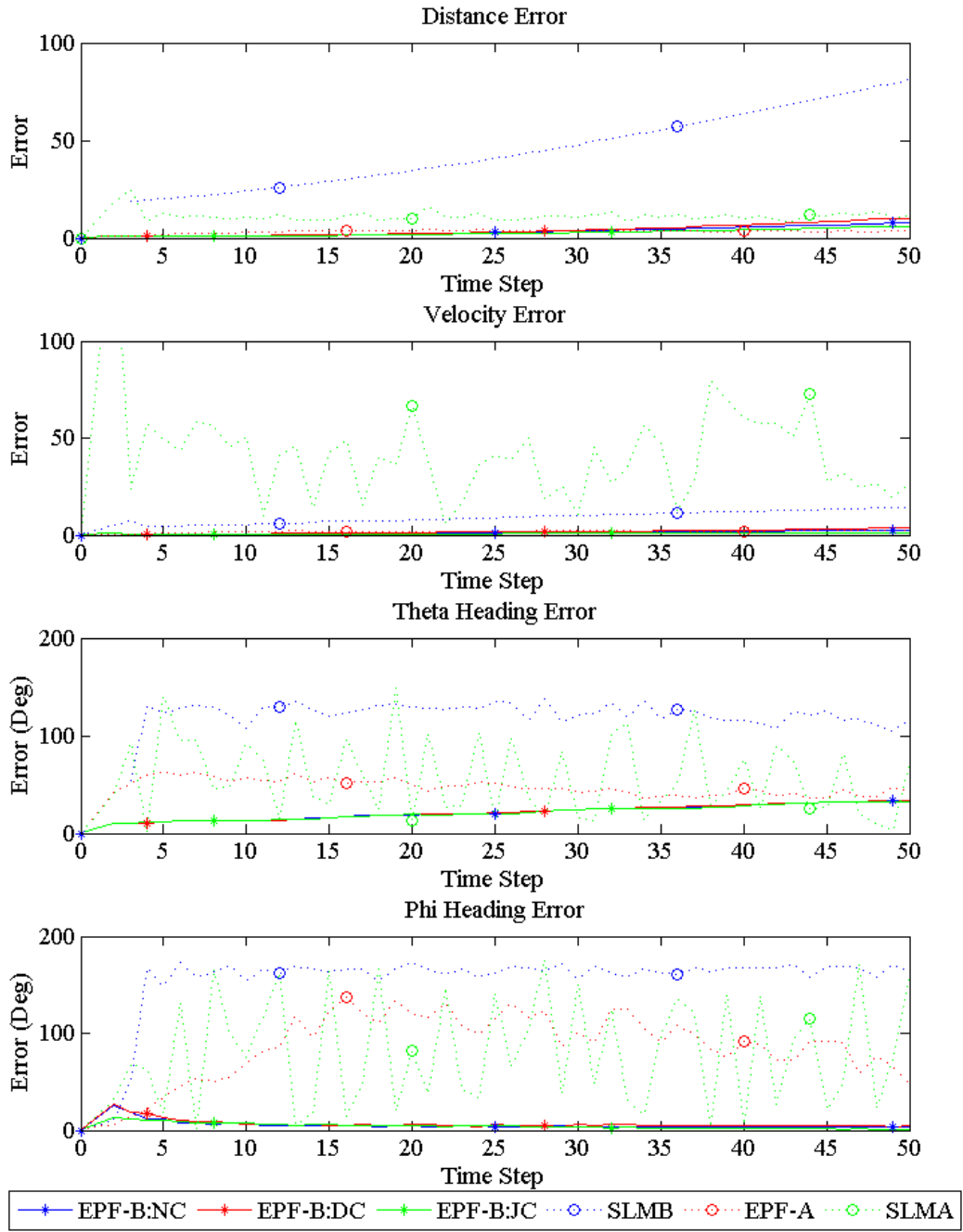


Figure 5.32: TER for Equal Weight Matrices with Noise

With the addition of noise, all three variations of EPF-B decreased their error compared to the simulations without noise. Though seemingly paradoxical, the noise allows the filter to select from a larger spread of particles. This appears to mitigate the lagging issue caused by insufficient variance for the predicted states. Additionally, the one filter that could change its variance, the JC becomes locked on incorrect ϕ heading values. When the JC determines the velocity is increasing, it assigns additional variance to the velocity prediction at the expense of the heading angles. If the variance needed to predict a state decreases to the point that the spread is insufficient to allow the particle to move towards the observed state, then a lock may occur. This does not lead to weight collapse however, since the filter can still use measurements for other variables.

5.4.2 Unequal Matrix.

The use of an unequal matrix attempts to address the difficulties in determining depth. By increasing the weights assigned to the three velocities, s , \dot{u}_p , and \dot{v}_p , it may be possible to more accurately determine depth if the filter prefers accurate measurements for these states. The relationship between the velocities and depth is seen in Equation 4.23. Figure 5.33 and 5.34 show the effects of the unequal weighing matrix without measurement noise.

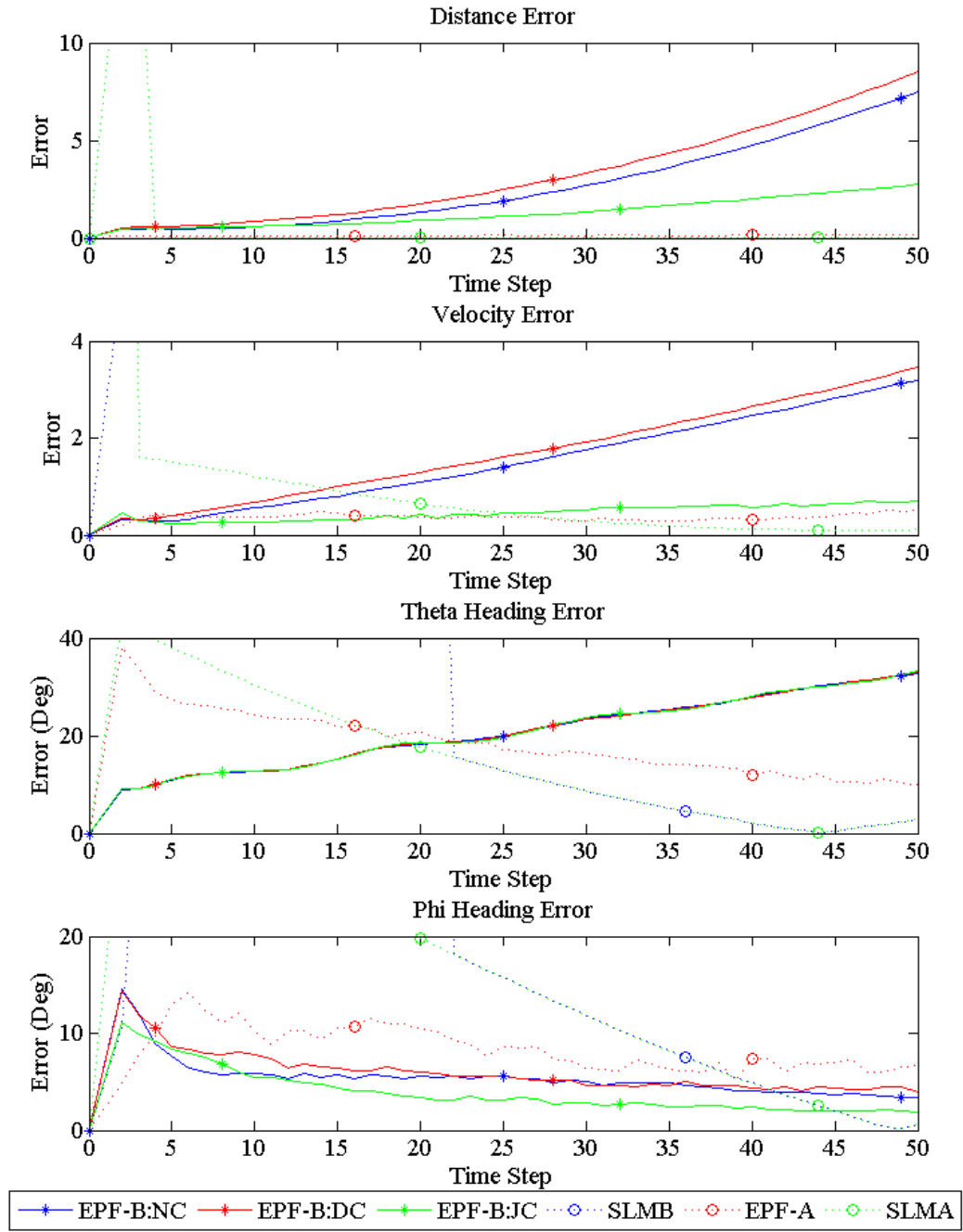


Figure 5.33: MAE for Unequal Weight Matrices without Noise

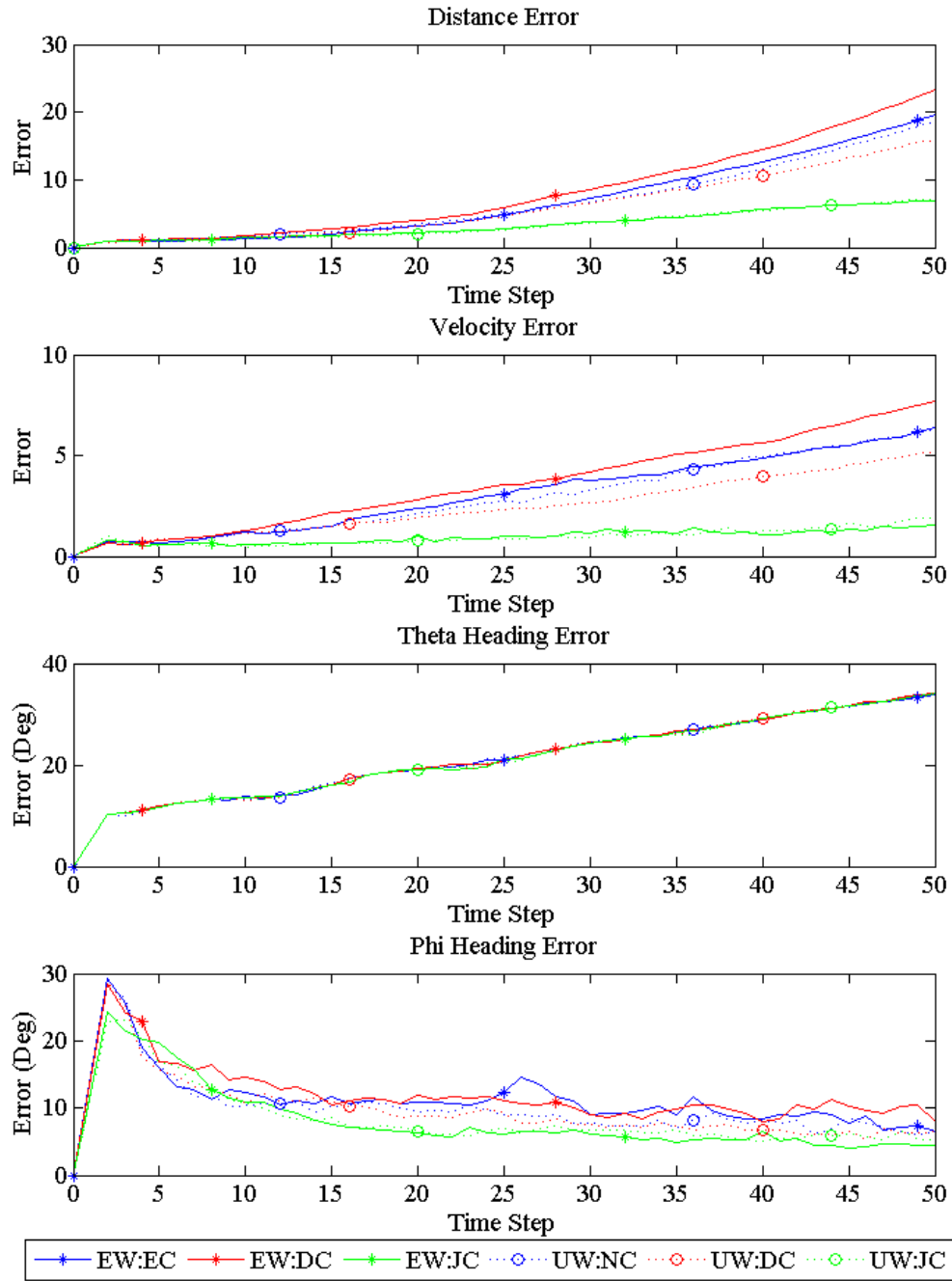


Figure 5.34: TER for Unequal Weight Matrices without Noise

The effects of the the unequal weighted matrix are limited for non-noisy measurements. There is slight improvement for velocity, and consequently distance, errors for DC. There was little to no improvement for the un-compensated or JC filters. Effects of the unequal weighted matrix with measurement noise are shown in Figure 5.35 and 5.36.

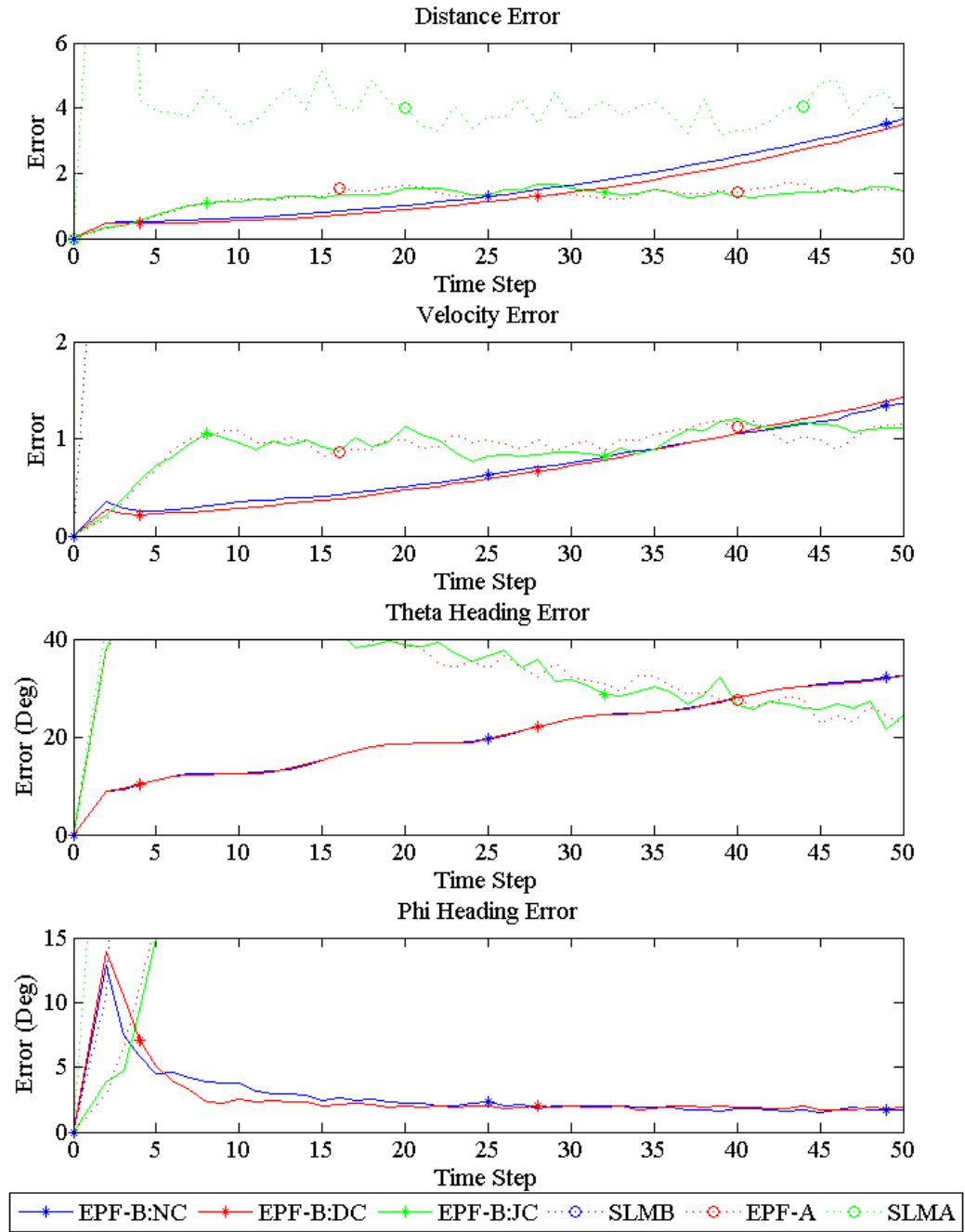


Figure 5.35: MAE for Unequal Weight Matrices with Noise

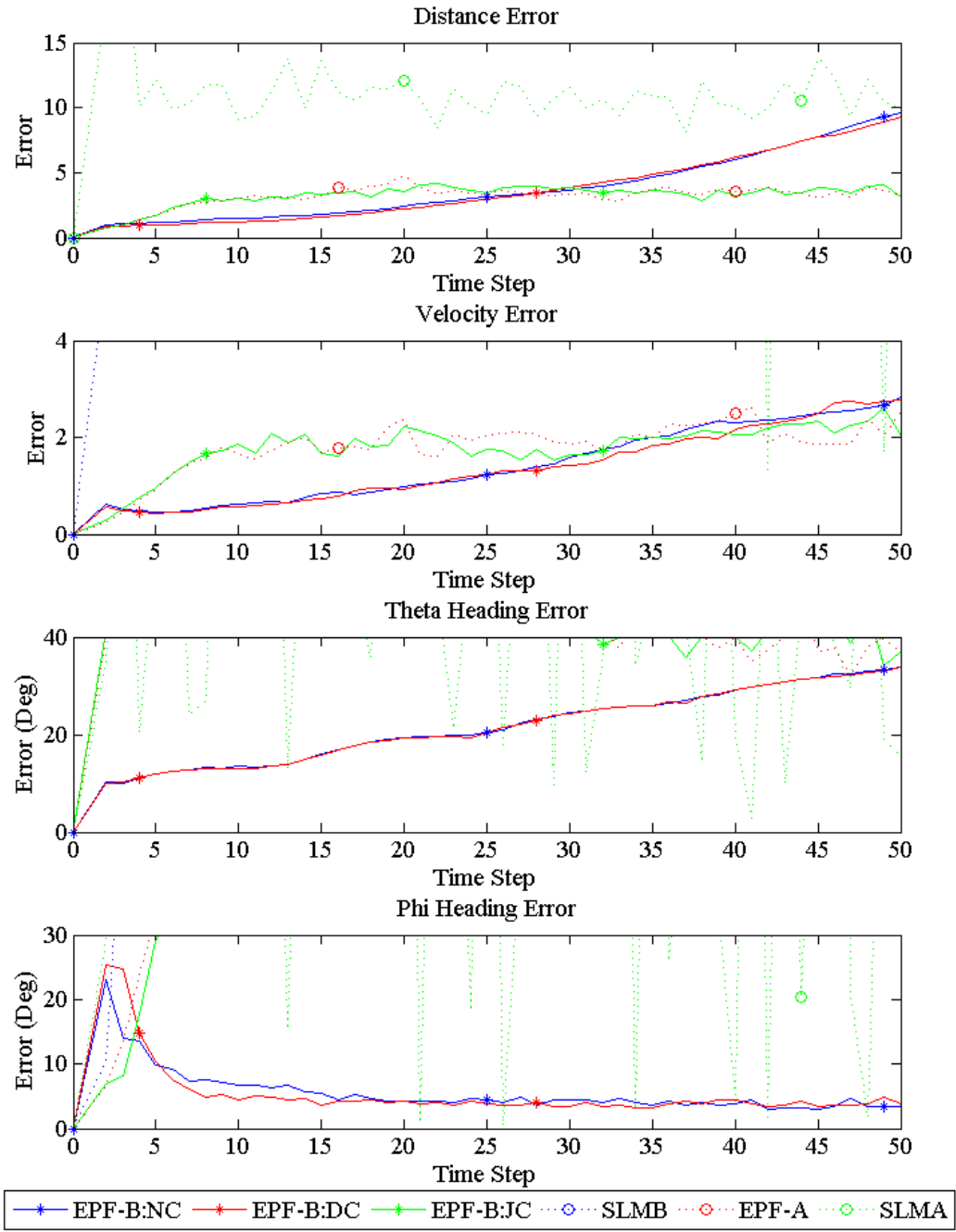


Figure 5.36: TER for Unequal Weight Matrices with Noise

The addition of noise had limited effects on the EPF-B compensator variations. Again, there was little discernible difference between the equal weighted and unequal weighted results for the uncompensated or JC filters. Figure 5.37 illustrates that there does not appear to be any discernible depth improvement between equally and unequally weighing matrices. In contrast to previous plots of Euclidean distance, Figure 5.33, 5.34, 5.35, and 5.36, Figure 5.37 only shows the depth. Any improvement in the Euclidian distance is due to better estimates of u_p and u_v , not depth, since all variants of EPF-B returned nearly equivalent erroneous depth estimates. Additionally, there does not appear to be any significant difference between the three EPF-B filters. The MAE and TER values are nearly identical since the successful simulations of the filter varied little from each other. Both MAE and TER decreased for EPF-B with the addition of noise, compared to simulations of EPF-B without noise. As discussed in Section 5.4.1, the difference between simulations with noise and those without is not due to better filter performance, but rather lower filter tolerance. The results improved since EPF-B would only accept more accurate particle distributions else EPF-B would suffer weight collapse and crash until it received a more accurate particle distribution. A comparison of the weight collapses is detailed in Table 5.21.

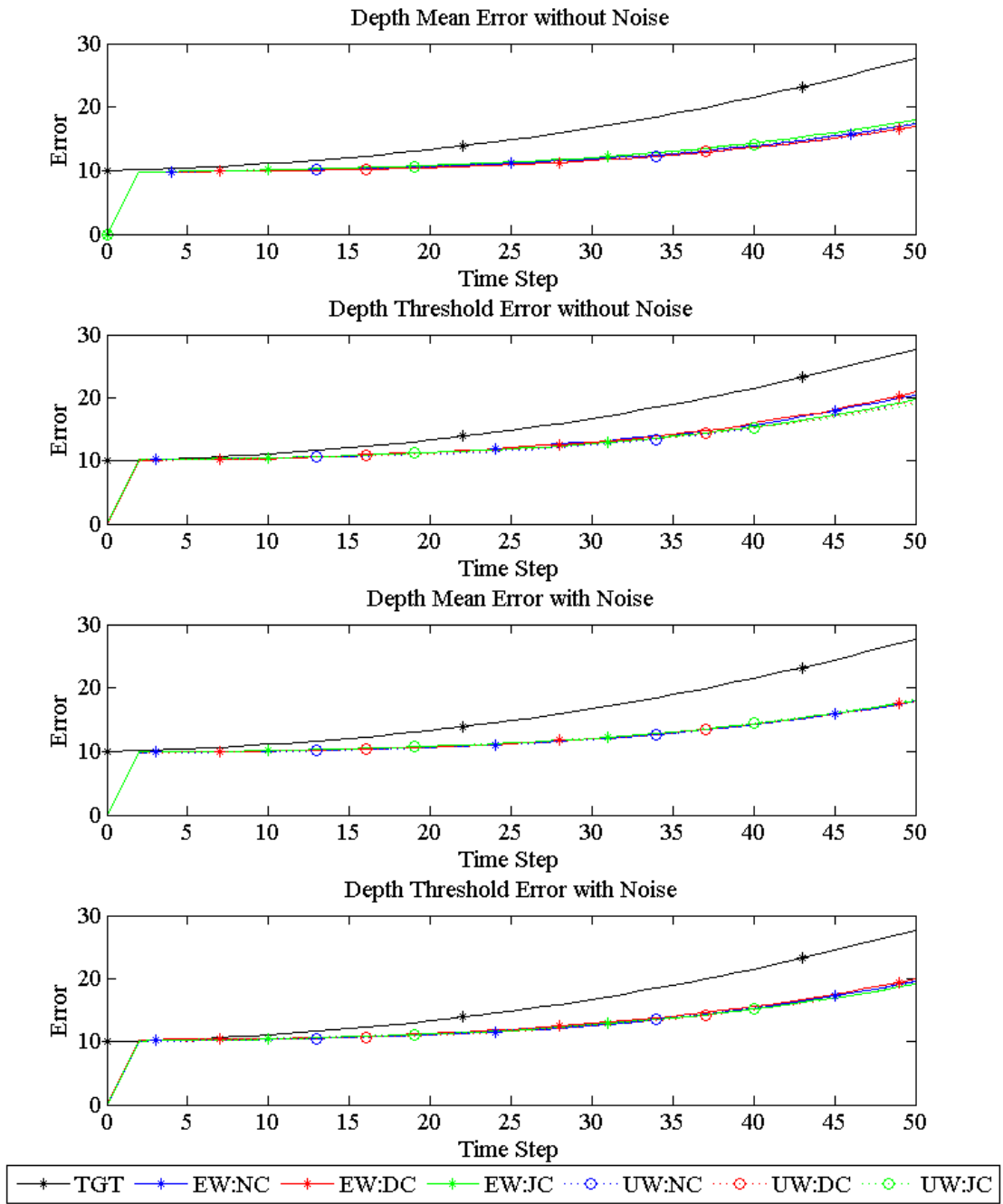


Figure 5.37: Mean and Threshold Depth Error

Table 5.19 and 5.20 detail the MAE and TER means for the variations of EPF-B as well as EPF-A, SLMB, and SLMA, without and with measurement noise respectively. Since all variants of EPF-B begin to diverge after 25 time-steps while EPF-A and the linear filters converge after 25 time-steps, different ranges were used to calculate the mean values as shown in Table 5.18. This was done to provide a more accurate comparison.

Table 5.18: EPF-B Time Unit Ranges for Mean Error

Filter	Time Range
EPF-B:NC	0-25
EPF-B:DC	0-25
EPF-B:JC	0-25
EPF-A:	25-50
SLMB:	25-50
SLMA:	25-50

Table 5.19: EPF-B and Comparison Mean Errors for Measurements without Noise

Metrics	D	V	θ	ϕ
Mean Absolute Error				
EPF-B: EW, NC	4.2604	2.2522	26.4100	4.4401
EPF-B: EW, DC	5.0248	2.4407	26.3926	4.6636
EPF-B: EW, JC	1.8196	0.5765	26.4099	2.4577
EPF-B: UW, NC	4.3671	2.1822	26.4194	3.8628
EPF-B: UW, DC	4.2236	2.0363	26.4469	3.6290
EPF-B: UW, JC	2.0992	0.7304	26.3935	2.8901
EPF-A:	0.1466	0.3619	13.5314	6.8414
SLMB:	47.0796	12.2979	4.8728	6.9306
SLMA:	0.0214	0.1972	4.8728	6.9310
Threshold Error				
EPF-B: EW, NC	11.3292	4.5755	27.5466	9.5354
EPF-B: EW, DC	13.2298	5.3592	27.5452	9.7998
EPF-B: EW, JC	4.8822	1.2041	27.4622	5.3966
EPF-B: UW, NC	10.5499	4.3951	27.5951	7.8076
EPF-B: UW, DC	9.6324	3.5654	27.5577	7.1292
EPF-B: UW, JC	4.9113	1.2384	27.3523	6.1313
EPF-A:	0.3051	0.7234	17.8481	14.1513
SLMB:	59.9637	12.2979	4.8728	6.9306
SLMA:	0.0272	0.1972	4.8728	6.9310

Table 5.20: EPF-B and Comparison Mean Errors for Measurements with Noise

Metrics	D	V	θ	ϕ
Mean Absolute Error				
EPF-B: EW, NC	2.2233	0.9609	26.3606	1.6955
EPF-B: EW, DC	2.5175	1.1234	26.3990	2.1413
EPF-B: EW, JC	1.5520	0.4610	26.3718	1.1535
EPF-B: UW, NC	2.3045	0.9605	26.3809	1.8514
EPF-B: UW, DC	2.0916	0.9660	26.3437	1.8992
EPF-B: UW, JC	1.4299	0.9968	29.2395	41.2272
EPF-A:	1.4976	1.1013	28.7480	40.4263
SLMB:	46.9995	10.9148	64.3976	92.0850
SLMA:	3.8670	40.0793	52.7010	81.2926
Threshold Error				
EPF-B: EW, NC	5.0777	1.7264	27.3585	3.5118
EPF-B: EW, DC	6.0791	2.3830	27.5268	4.8032
EPF-B: EW, JC	3.8818	0.8895	26.9046	2.4923
EPF-B: UW, NC	5.6134	2.0294	27.3875	3.9196
EPF-B: UW, DC	5.6078	1.9531	27.3141	3.8214
EPF-B: UW, JC	3.5946	1.9848	42.3245	90.2125
EPF-A:	3.6158	2.1992	41.6683	93.1232
SLMB:	59.9568	11.6336	121.0205	165.1795
SLMA:	10.9642	40.0239	49.7049	87.6773

An additional metric used is the number of weight collapses that occurred for each particle filter per number of simulations. During each scenario, the number of times a filter crashed due to weight collapse was recorded and the simulation was restarted at time $t = 0$. This process was repeated until the filter completed all simulations in the scenario successfully. This total number of crashes was divided by the number of simulations used in that scenario. Thus, the metric can be interpreted as the number of weight collapses that will occur, on average, for each successful simulation. It serves as a measure of stability. Neither of the linear filters, SLMA and SLMB, use particles and thus do not have a weight collapse metric.

Table 5.21: Weight Collapse for Measurements without and with Noise

	Without Noise	With Noise
EPF-B: Equal Weight, No Comp	16.08	86.30
EPF-B: Equal Weight, Depth	12.82	67.02
EPF-B: Equal Weight, Jacobain	158.82	806.46
EPF-B: Unequal Weight, No Comp	12.56	77.86
EPF-B: Unequal Weight, Depth	24.32	50.48
EPF-B: Unequal Weight, Jacobain	122.86	771.20
EPF-A:	0	0

As seen in Table 5.21, EPF-A never suffered a weight collapse for any of its simulations. The variants of EPF-B suffered additional weight collapse with the introduction of noise, which is logical since it is more difficult to determine the hidden states in the presence of noise. However, due to the high rate of collapse, none of the versions of EPF-A are mature enough to be used without additional development to stem the weight collapse. The JC suffered the most frequent weight collapse which indicates it

is the most sensitive variant of EPF-B. Additionally, the high rate of weight collapse for JC indicates the filter is not determining the best variance values to use.

5.4.3 Evaluated Particle Filter B Summary.

Based on the results from the EPF-B evaluation scenarios, additional development is required to make the filter viable. All variants diverged to some degree, however the JC did so the least, indicating the possibility that positive tracking could be achieved. However, the JC is also the least stable filter by a significant factor. Further investigation is required to determine the most appropriate method to change the variance.

VI. Conclusions and Future Work

6.1 Conclusion

This thesis developed and evaluated the possibility of using a particle filter to track non-linear targets with both 3-D position data and pixel data from a single camera or observer without accurate depth information. In order to track non-linear targets, the ability of the particle filter to track non-linear targets depends on its ability to determine hidden states, velocity and headings, using both clean as well as noisy state variables and measurements. Additionally, the particle filter used nonlinear models as its basis. Simulations of vision based systems were developed to evaluate the particle filter for simulated measurements from a system with accurate depth measurements and one without. Two distinct particle filters were developed, for each respective system, EPF-A and EPF-B. In conjunction with the particle filters, two linear models were also developed, SLMA and SLMB, and a Kalman filter, EKF-A, to serve as a baseline comparisons for the particle filters' performance.

EPF-A's performance was evaluated using a variety of tests. Two series of tests constrained the movement of the target, thus using a degenerate form of the model present in EPF-A, in order to evaluate the filter's ability to track a target that behaved according to a subset of the model used in EPF-A. The filter tracked the target accurately for all hidden states while SLMA could only track the position states that directly correlated to the measurements of the target. With the addition of noise, EPF-A could still track the hidden states, albeit with slightly less accuracy. However, SLMA could not track any of the states. The final series of tests did not constrain the target model, and included constant acceleration. EPF-A closely tracked the target, despite the presence of noise, while the SLMA could not track the target, including its position. EKF-A could track the target, however its hidden state estimations were less accurate than those of EPF-A. This test

demonstrated that the filter is well-suited to track targets with noisy measurements when the filter uses an accurate model to generate its predictions.

EPF-B's performance was also evaluated, though only a single scenario was used due to EPF-B's propensity towards weight collapse. Additional scenarios were deemed unnecessary until the weight collapse concern is addressed. A variety of methods were evaluated to determine if any could reduce weight collapse. Although some improved EPF-B's performance marginally, none could successfully mitigate the weight collapse issue. This is a key area for improvement, as illustrated by Table 5.21. Additionally, the sensitivity of the filter to measurements prevents it from being used as effectively as EPF-A to find the target. Rather, EPF-B must start with the same initial conditions as the target in order to track it.

6.2 Practical Considerations

The particle filter, being probabilistic, has several key distinguishing features from traditional linear filters that must be considered by the user.

1. **Filter Model** - First and foremost, the filter model should match the target as closely as possible. If the filter model begins to differ from the target, then the particles it produces begin to differ from the measurements, consequently producing erroneous predictions. Additionally, since the filter does not require a linear model, no effort should be made to simplify the model unless due to computational concerns (in which case the particle filter may not be the most suitable filter anyway). Additionally, the model is what allows the filter to produce reasonable estimates despite the presence of noise. As noise variances increase, the filter relies more on its own predictions rather than the measurements. If these predictions are inaccurate due to a poor model, then the filter will eventually diverge from the target regardless of noise. Even with perfect measurements, if the model is inaccurate, the filter may not be able to determine the state variables of interest. One potential method of resolving

these model inaccuracies is to incorporate a machine learning phase into the filter and allow the filter to alter its own models based on the measurements.

2. State Variables - Regarding the state variables, only the highest order variables should be varied using their corresponding system variance. If lower order or measurement variables are varied, the particle filter may select particles due to the variance. While these particles may more closely match the measurements, they will no longer correspond with their respective higher order variables due to the additional variance 'contaminating' their value.
3. System Variances - Similarly to the model, system variances should match the target's system variances as close as possible. The system variances help define the target distribution from which particles are sampled. If the variances are too large, then the particles are spread further from the most likely states resulting in decreased fidelity. Additionally, large variances may result in multiple viable solutions with complex models, as demonstrated by EPF-B. When the system variance became too large, the particle filter could select measurements that resulted in incorrect state variables since multiple combinations of the state variables could yield the same measurements. If the variances are too tight, then the correct solutions may lie outside the distribution of particles. This too can cause the error between the target and filter states to grow since the filter does not have the ability to produce states equal to the target states. This phenomenon was also observed with EPF-B when the speed began to drift behind the target speed, causing a cascading ripple through the other lower order states, such as position. If left uncorrected, this can lead to weight collapse since the filter will eventually be unable to generate a prediction that is close enough to the measurements to generate a sufficient weight.

4. Dimensionality - Similarly to other filters, as the number of dimensions increases, the size of the state space must also increase to accommodate these dimensions. The likelihood of weight collapse increases as the number of dimensions increases, since the potential number of solutions may vastly exceed the number of particles. Thus, the correct solution may be ‘lost’ in the state-space. Adding more particles to model this state space is also a temporary solution; eventually the number of particles needed to accurately model the state space will exceed the computational capabilities at hand. Efforts to mitigate this vulnerability are still ongoing [5].
5. Measurement Noise - Analogous to Kalman gain, the measurement noise determines the degree to which the filter trusts the measurements. The larger this value, the more the filter will rely on the model predictions, once again illustrating the necessity of an accurate model. The smaller this value, the more the filter will trust the measurements.
6. Weighing Matrix - The weighing matrix is an additional means to control what the filter relies on for its measurements. The different weights correspond to how much the filter will rely on each measurement to generate the importance weight. Depending on the system, some measurements may be inherently less reliable than others, so their impact should be minimized, but not entirely eliminated since they do characterize the state variables to some degree. Also, if the user wishes to track a particular state with increased accuracy, the weights of the measurements that correspond to that state could be increased.
7. Weight Distribution - Although a normal distribution was used in this research to generate the importance weights, any type of distribution function may be used, so long as the distribution mirrors that target’s noise distribution.

8. Statistics - The end product of the particle filter is a discrete probability distribution. A variety of statistical methods may be used to generate the predicted state variables; a mean is one of the simplest, but more advanced methods will likely return more robust predictions.

6.3 Future Work

There are three key lines of future work concerning this research: real-world implementation, further development, and space-environment simulation. Although the EPF-A has been developed and tested within an simulated environment, it must be tested on an actual system in order to fully validate its functionality and performance. The most likely system is the PTZ camera setup developed at AFIT. Note, this current setup provides the depth measurement needed by EPF-A by using two PTZ cameras to provide stereo vision. The particle filter could be used in conjunction with the current system to provide more accurate global target locations. Ideally, the results from the particle filter would be used to direct the cameras to their next position, anticipating the movement of the target in order to provide smoother and more accurate tracking. One of the key benefits of the particle filter is that multiple non-linear target models could be used depending on the target's dynamic properties. Currently, the particle filter researched in this thesis is a relatively simple SIS. As demonstrated in Section 3.4.5.4 and discussed in Section 2.5.6, weight collapse remains an potential issue for EPF-A and a definite issue for EPF-B. More advanced filters could mitigate such collapses by both identifying potential collapses and redistributing the particles to stave off a collapse and by improving the tracking ability of the filter. Better tracking results in higher particle weights, thus again reducing the risk of weight collapse.

One such area of improvement that is unexplored is more advanced statistical methods to both better model the discrete posterior PDF produced by the particle filter as well as subsequently sampling this PDF to produce the set of final states. Improvements could

also be made to the variances; e.g. how they are generated and how they are updated as the particle filter tracks the target. This area was explored with the various compensators developed for EPF-B and although the results were not definite, additional exploration should be conducted. Although JC holds the most potential to estimate depth, due to its ability to alter the filter's variances, in its present form it is unsuitable due to its severe weight collapse concerns.

The final primary line of research focuses on using the particle filter to track space-objects based on limited measurements and subsequently generate orbit properties of that object. The particle filter performs ideally when the model used by the filter more closely resembles the target. The space environment is a unique environment, one that has been meticulously modeled and where models are able to predict accurate long-term behavior of objects. The particle filter could be used to observe objects and determine their hidden states, in this case orbital parameters, potentially using a small set of measurements. These measurements could be vision-based from satellites, and possibly taken single camera if the particle filter is able to render accurate depth information. By increasing the number of sensors, orbit conflicts can be mitigated, helping to reduce congestion within the space environment.

Appendix: MATLAB Code

A.1 Introduction

The following appendices contain the MATLAB code for the particle filters PPF-A, PPF-B, EPF-A, and EPF-B, as well as the Kalman filters and linear models, PKF-A, PKF-B, SLMA, and SLMB. Also included are the functions to generate the plots and metrics. The functions are grouped based on which particle filter is being evaluated.

A.2 Prototype Particle Filter A

A single function, PPF_A_Final.m, executes and plots PPF-A.

```
clear all; close all; clc;

%% PPF-A_Final

%This program will execute PPF-A which is a single dimension particle filter
%using a highly non-linear function to demonstrate the particle filter's
%ability to track non-linear functions. No comparison filter is provided
%due to the highly non-linear function.

%% Defaults

% Change default axes fonts

set(0, 'DefaultAxesFontName', 'Times New Roman')
set(0, 'DefaultAxesFontSize', 14)

% Change default text fonts

set(0, 'DefaultTextFontName', 'Times New Roman')
set(0, 'DefaultTextFontSize', 14)

%% Initialize Variables

%Initial target variables

    %Total number of time steps (run time)
```

```

T=70;

t=1;

%Initial state
x(1)=0.1;

%Process/system noise covariance
x_N=1;

%Measurement noise covaraiance
x_R=1;

%Initial observations
%Target measurement
y(1) = [x(1)^2 / 20 + sqrt(x_R)*randn];

%Initial particle variables
%Number of particles
N=100;

%Variance of initial estimate
V=2;

%Initialize particles based on initial conditions (create first
%distribution of particles)
for i = 1:N
    x_P(1,i) = x + sqrt(V)*randn;
end

%% Begin simulating
for t = 2:T
    %% Truth Model
    %Propogate state forward
    x(t)=0.5*x(t-1)+25*x(t-1)/(1+x(t-1)^2)+8*cos(1.2*(t-1))+sqrt(x_N)*randn;
    %Propogate measurements forward based on states

```

```

y(t)=x(t)^2/20 + sqrt(x_R)*rand;

%% Particle Filter
for i=1:N
    %Update states for model
    x_P(t,i)=.5*x_P(t-1,i)+25*x_P(t-1,i)/(1+x_P(t-1,i)^2)+...
        8*cos(1.2*(t-1))+sqrt(x_N)*randn;
    %Measurement update
    y_P(t,i)=x_P(t,i)^2/20;
    %Generate and assign importance weights to particles
    P_w(t,i)=(1/sqrt(2*pi*x_R))*exp(-(y(t)-y_P(t,i))^2/(2*x_R));
end

%Normalize to form a probability distribution
P_w(t,:) = P_w(t,:)./sum(P_w(t,:));

%Resample particles to form new distribution
%What this code specifically does is randomly, uniformly, sample from
%the cummulative distribution of the probability distribution
%generated by the weighted vector P_w. If you sample randomly over
%this distribution, you will select values based upon there statistical
%probability, and thus, on average, pick values with the higher weights
%(i.e. high probability of being correct given the observation z).
%store this new value to the new estimate which will go back into the
%next iteration
for i=1:N
    P_get(t,i)=find(rand<=cumsum(P_w(t,:)),1);
    x_P(t,i)=x_P(t,P_get(t,i));
end

x_est(t) = mean(x_P(t,:));
end

```

```

t=1:T;
hfig=figure('name','PPF-A');
set(hfig,'Position',[100,100,800,400]);
plot(t, x, '*-b', t, x_est, '*-r', t, y, '*--g');
    xlabel('Time Step'); ylabel('Position');
    legend('Target', 'PPF-A','Measurement');
    set(legend,'Orientation','horizontal','Location','SouthOutside');

```

A.3 Prototype Particle Filter B

A single function, PPF_B_Final.m, executes and plots PPF-B, PKF-A, and PKF-B.

```

clear all; close all; clc;
%% PPF-A_Final
%This program will execute PPF-A which is a single dimension particle filter
%using a highly non-linear function to demonstrate the particle filter's
%ability to track non-linear functions. No comparison filter is provided
%due to the highly non-linear function.
%% Defaults
% Change default axes fonts
set(0,'DefaultAxesFontName','Times New Roman')
set(0,'DefaultAxesFontSize', 14)

% Change default text fonts
set(0,'DefaultTextFontName', 'Times New Roman')
set(0,'DefaultTextFontSize', 14)

%% Initialize Variables
%Initial target variables
    %Total number of time steps (run time)
    T=70;

```

```

t=1;

%Initial state
x(1)=0.1;

%Process/system noise covariance
x_N=1;

%Measurement noise covaraiance
x_R=1;

%Initial observations
%Target measurement
y(1) = [x(1)^2 / 20 + sqrt(x_R)*randn];

%Initial particle variables
%Number of particles
N=100;

%Variance of initial estimate
V=2;

%Initialize particles based on initial conditions (create first
%distribution of particles)
for i = 1:N
    x_P(1,i) = x + sqrt(V)*randn;
end

%% Begin simulating
for t = 2:T
    %% Truth Model
    %Propogate state forward
    x(t)=0.5*x(t-1)+25*x(t-1)/(1+x(t-1)^2)+8*cos(1.2*(t-1))+sqrt(x_N)*randn;
    %Propogate measurements forward based on states
    y(t)=x(t)^2/20 + sqrt(x_R)*rand;

```

```

%% Particle Filter

for i=1:N

    %Update states for model

    x_P(t,i)=.5*x_P(t-1,i)+25*x_P(t-1,i)/(1+x_P(t-1,i)^2)+...
        8*cos(1.2*(t-1))+sqrt(x_N)*randn;

    %Measurement update

    y_P(t,i)=x_P(t,i)^2/20;

    %Generate and assign importance weights to particles

    P_w(t,i)=(1/sqrt(2*pi*x_R))*exp(-(y(t)-y_P(t,i))^2/(2*x_R));

end

%Normalize to form a probability distribution

P_w(t,:) = P_w(t,:)./sum(P_w(t,:));

%Resample particles to form new distribution

%What this code specifically does is randomly, uniformly, sample from
%the cumulative distribution of the probability distribution
%generated by the weighted vector P_w. If you sample randomly over
%this distribution, you will select values based upon there statistical
%probability, and thus, on average, pick values with the higher weights
%(i.e. high probability of being correct given the observation z).
%store this new value to the new estimate which will go back into the
%next iteration

for i=1:N

    P_get(t,i)=find(rand<=cumsum(P_w(t,:)),1);

    x_P(t,i)=x_P(t,P_get(t,i));

end

x_est(t) = mean(x_P(t,:));

end

```

```

t=1:T;
hfig=figure('name','PPF-A');
set(hfig,'Position',[100,100,800,400]);
plot(t, x, '*-b', t, x_est, '*-r', t, y, '*--g');
    xlabel('Time Step'); ylabel('Position');
    legend('Target', 'PPF-A','Measurement');
    set(legend,'Orientation','horizontal','Location','SouthOutside');

```

A.4 Evaluated Particle Filter A

Two functions executed and plotted EPF-A, EKF-A, and SLMA. The first function, EPF_A_Execute_Final.m, provides the initial conditions to the second function, EPF_A_Function_Final.m, which executes the filters.

A.4.1 EPF A Execute Final.

```

%% This script analyzes EPF-A
clear all; close all; clc;

%% Defaults
% Change default axes fonts
set(0,'DefaultAxesFontName','Times New Roman')
set(0,'DefaultAxesFontSize', 14)

% Change default text fonts
set(0,'DefaultTextFontName', 'Times New Roman')
set(0,'DefaultTextFontSize', 14)

%% Parameters
%Number of simulation runs
sim_run=1;
sim_plot=sim_run;
%Threshold Error Range

```

```

solution_range = .9;

%Iteration Number
params(1)      =    150;

%Time Steps
params(2)      =    .1;
dt=params(2);
T=params(1);

%% Target Parameters
%Straight Line, x-axis: 1
%Straight Line, y-axis: 2
%Straight Line, z-axis: 3
%Straight Line, -x-axis: 4
%Straight Line, -y-axis: 5
%Straight Line, -z-axis: 6
%Circle, x/y, about z: 7
%Circle, x/z, about y: 8
%Circle, y/z, about x: 9
%Prelim, chap 5:      10
%MANUAL, see below:   0
target_scenario =0;

%Process Variance
%0.1:    2
%None:   0
P_variance_scenario = 2;

%No measurment variance:    1
%5 Measurement variance:    2
%10 Measurement variance:   3
%MANUAL:                     0
m_variance_scenario =2;

```

```

%% Kalman Filter Parameters

%Initial Conditions
%Match target ICs: 1
%Without, at 0: 2
Kalman-ICs = 1;

%Control Law
%Axial Simulation: 1
%Circular Simulation: 2
%Random Simulation: 3
%Set to 0: 0
Kalman_Control_Law = 1;

%Process Variance
Kalman_Process_Variance = 1;

%Measurement Variance
Kalman_Measurement_Variance = 1;

%% EPF-A Parameters
%With target intial conditions: 1
%Without, at 0 (x,y,z): 2
%For circle, x/y about z: 3
%For circle, x/z about y: 3
%For circle, y/z about y: 3
%MANUAL: 0
filter1_scenario = 2;

%Number of particles
p_num = 500;

```

```

%Weights
%All equal: 1
%MANUAL:      0
filter1_weight =1;

%Filter variances
%Scenario 1:    1
%Standard:      2
%MANUAL:        0
filter1_variance =2;

%Measurment Variance
%Match measurment: 1
%Variance of .01: 2
%MANUAL:        0
%NOTE: CANNOT BE 0
filter1_measurement_variance = 1;

%Manual values
%Target
manual(1) = 5; %x
manual(2) = 5; %y
manual(3) = 5; %z
manual(4) = 4; %V
manual(5) = 45; %theta
manual(6) = 45; %phi
manual(7) = 0; %dx
manual(8) = 0; %dy
manual(9) = 0; %dz
manual(10) = .1; %dV
manual(11) = 5; %dtheta
manual(12) = 5; %dphi

```

```

manual(13) = 0; %x_dot
manual(14) = 0; %y_dot
manual(15) = 0; %z_dot

manual(16) = 0; %u measurement variance
manual(17) = 0; %v measurement variance
manual(18) = 0; %w measurement variance

%EPF-A
manual(19) = 0; %x
manual(20) = 0; %y
manual(21) = 0; %z
manual(22) = 0; %V
manual(23) = 0; %theta
manual(24) = 0; %phi
manual(25) = 0; %dx
manual(26) = 0; %dy
manual(27) = 0; %dz
manual(28) = 0; %dV
manual(29) = 0; %dtheta
manual(30) = 0; %dphi

manual(31) = 0; %p_num

manual(32) = 0; %x variance
manual(33) = 0; %y variance
manual(34) = 0; %z variance
manual(35) = 0; %V variance
manual(36) = 0; %theta variance
manual(37) = 0; %phi variance
manual(38) = 0; %dvp variance

```

```

manual(39) = 0; %dyp variance
manual(40) = 0; %dzp variance
manual(41) = 0; %dVp variance
manual(42) = 0; %dthetap variance
manual(43) = 0; %dphip variance

manual(44) = 0; %Measurement noise covariance

manual(45) = 1; %Filter1 u weight
manual(46) = 1; %Filter1 v weight
manual(47) = 1; %Filter1 w weight

switch target_scenario
case 1
    x=1; y=0; z=0; V=4; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 2
    x=0; y=1; z=0; V=4; theta=0; phi=0; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 3
    x=0; y=0; z=1; V=4; theta=90; phi=90; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 4
    x=1; y=0; z=0; V=4; theta=90; phi=180; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 5
    x=0; y=1; z=0; V=4; theta=180; phi=0; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 6
    x=0; y=0; z=5; V=4; theta=90; phi=270; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;

```

```

case 7
    x=0; y=0; z=10; V=5; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
    dtheta=1; dphi=0; xdot=0; ydot=0; zdot=0;
case 8
    x=0; y=0; z=10; V=5; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
    dtheta=0; dphi=1; xdot=0; ydot=0; zdot=0;
case 9
    x=0; y=0; z=10; V=5; theta=0; phi=90; dx=0; dy=0; dz=0; dV=0;...
    dtheta=1; dphi=0; xdot=0; ydot=0; zdot=0;
case 10
    x=5; y=5; z=5; V=1; theta=45; phi=45; dx=0; dy=0; dz=0; dV=.1;...
    dtheta=5; dphi=5; xdot=0; ydot=0; zdot=0;
case 0
    x>manual(1); y>manual(2); z>manual(3); V>manual(4);...
    theta>manual(5);
    phi>manual(6); dx>manual(7); dy>manual(8); dz>manual(9);...
    dV>manual(10);
    dtheta>manual(11); dphi>manual(12); xdot>manual(13);...
    ydot>manual(14); zdot>manual(15);
otherwise
    error('INCORRECT TARGET SCENARIO SELECTION')
end

switch P_variance_scenario
case 1
    TProcess_V=.1; TProcess_theta=deg2rad(5); TProcess_phi=deg2rad(5);
case 2
    TProcess_V=.01; TProcess_theta=.01; TProcess_phi=.01;
case 0
    TProcess_V=0; TProcess_theta=0; TProcess_phi=0;
otherwise
    error('INCORRECT PROCESS VARIANCE SCENARIO SELECTION')

```

end

switch m_variance_scenario

case 1

Mu_var=0; Mv_var=0; Mw_var=0;

case 2

Mu_var=5; Mv_var=5; Mw_var=5;

case 3

Mu_var=10; Mv_var=10; Mw_var=10;

case 0

Mu_var>manual(16); Mv_var>manual(17); Mw_var>manual(18);

otherwise

error('INCORRECT VARIANCE SCENARIO SELECTION')

end

switch Kalman_ICs

case 1

xk1=x; yk1=y; zk1=z; Vk1=V; thetak1=theta; phik1=phi;

dxk1=dx; dyk1=dy; dzk1=dz; dVk1=dV; dthetak1=dtheta; dphik1=dphi;

case 2

xk1=0; yk1=0; zk1=0; Vk1=0; thetak1=0; phik1=0;

end

switch Kalman_Control_Law

case 1

K_ux=(dV/dt)*sind(thetak1)*cosd(phik1)*(dt^2/2)+...

(dtheta/dt)*Vk1*cosd(thetak1)*cosd(phik1)*(dt^2/2)+...

-(dphi/dt)*Vk1*sind(thetak1)*sind(phik1)*(dt^2/2);

K_uy=(dV/dt)*cosd(thetak1)*(dt^2/2)-...

(dtheta/dt)*Vk1*sind(thetak1)*(dt^2/2);

```

        K_uz=(dV/dt)*sind(thetak1)*sind(phik1)*(dt^2/2)+...
            (dtheta/dt)*Vk1*cosd(thetak1)*sind(phik1)*(dt^2/2)+...
            (dphi/dt)*Vk1*sind(thetak1)*cosd(phik1)*(dt^2/2);

    case 2
        K_ux=.5; K_uy=.5; K_uz=.5;
    case 3
        K_ux=1; K_uy=1; K_uz=1;
end

switch Kalman_Process_Variance
    case 1
        KProcess_x=1; KProcess_y=1; KProcess_z=1;
        KProcess_dx=TProcess_V*sind(TProcess_theta)*cosd(TProcess_phi)*dt;
        KProcess_dy=TProcess_V*cosd(TProcess_theta)*dt;
        KProcess_dz=TProcess_V*sind(TProcess_theta)*sind(TProcess_phi)*dt;
    end

switch Kalman_Measurement_Variance
    case 1
        KMu_var=Mu_var; KMv_var=Mv_var; KMw_var=Mw_var;
    end

switch filter1_scenario
    case 1
        xp1=x; yp1=y; zp1=z; Vp1=V; thetap1=theta; phip1=phi;
        dxp1=dx; dyp1=dy; dzp1=dz; dVp1=dV; dthetap1=dtheta; dhip1=dphi;
    case 2
        xp1=0; yp1=0; zp1=0; Vp1=0; thetap1=0; phip1=0;
        dxp1=0; dyp1=0; dzp1=0; dVp1=0; dthetap1=0; dhip1=0;
    case 3

```

```

    xpl=0; ypl=0; zpl=15; Vp1=0; thetap1=0; phip1=0;
    dxpl=0; dyp1=0; dzpl=0; dVp1=0; dthetap1=0; dhip1=0;
case 0
    xpl>manual(19); ypl>manual(20); zpl>manual(21);
    Vp1>manual(22); thetap1>manual(23); phip1>manual(24);
    dxpl>manual(25); dyp1>manual(26); dzpl>manual(27);
    dVp1>manual(28); dthetap1>manual(29); dhip1>manual(30);
otherwise
    error('INCORRECT FILTER1 INTIAL CONDITIONS')
end

switch filter1_variance
case 1
    xpl_v=1; ypl_v=1; zpl_v=1; Vp1_v=.5; thetap1_v=.1; phip1_v=.1;
    dxpl_v=.5; dyp1_v=.5; dzpl_v=.5; dVp1_v=.3; dthetap1_v=.1;...
        dhip1_v=.1;
case 2
    xpl_v=1; ypl_v=1; zpl_v=1; Vp1_v=1; thetap1_v=.1; phip1_v=.1;
    dxpl_v=.5; dyp1_v=.5; dzpl_v=.5; dVp1_v=.1; dthetap1_v=.01;...
        dhip1_v=.01;
case 0
    xpl_v>manual(32); ypl_v>manual(33); zpl_v>manual(34);
    Vp1_v>manual(35); thetap1_v>manual(36); phip1_v>manual(37);
    dxpl_v>manual(38); dyp1_v>manual(39); dzpl_v>manual(40);
    dVp1_v>manual(41); dthetap1_v>manual(42); dhip1_v>manual(43);
otherwise
    error('INCORRECT FILTER1 STATE VARIANCES')
end

switch filter1_measurement_variance
case 1
    m1_v=Mu_var;

```

```

case 2
    m1_v=.1;
case 3
    m1_v=.1;
case 0
    m1_v>manual(44);
otherwise
    error('INCORRECT FILTER1 MEASUREMENT VARIANCES')
end

```

```

switch filter1_weight
case 1
    W_u=1; W_v=1; W_w=1;
case 0
    W_u>manual(45); W_v>manual(46); W_w>manual(47);
otherwise
    error('INCORRECT FILTER1 WEIGHTS')
end

```

```

target(1) = x; %x
target(2) = y; %y
target(3) = z; %z
target(4) = V; %V
target(5) = theta; %theta
target(6) = phi; %phi
target(7) = dx; %dx
target(8) = dy; %dy
target(9) = dz; %dz
target(10) = dV; %dV
target(11) = dtheta; %dtheta
target(12) = dphi; %dphi
target(13) = xdot; %x_dot

```

```

target(14) = ydot; %y_dot
target(15) = zdot; %z_dot
target(16) = TProcess_V;
target(17) = TProcess_theta;
target(18) = TProcess_phi;

```

```

obs(1) = Mu_var;
obs(2) = Mv_var;
obs(3) = Mw_var;

```

```

Kfilter(1) = xk1;
Kfilter(2) = yk1;
Kfilter(3) = zk1;
Kfilter(4) = Vk1;
Kfilter(5) = thetak1;
Kfilter(6) = phik1;
Kfilter(7) = K_ux;
Kfilter(8) = K_uy;
Kfilter(9) = K_uz;
Kfilter(10) = KProcess_x;
Kfilter(11) = KProcess_y;
Kfilter(12) = KProcess_z;
Kfilter(13) = KProcess_dx;
Kfilter(14) = KProcess_dy;
Kfilter(15) = KProcess_dz;
Kfilter(16) = KMu_var;
Kfilter(17) = KMv_var;
Kfilter(18) = KMw_var;

```

```

filter1(1) = xp1; %x
filter1(2) = yp1; %y
filter1(3) = zp1; %z

```

```

filter1(4) = Vp1; %V
filter1(5) = thetap1; %
filter1(6) = phip1;
filter1(7) = dxp1;
filter1(8) = dyp1;
filter1(9) = dzp1;
filter1(10) = dVp1;
filter1(11) = dthetap1;
filter1(12) = dhip1;

filter1(13) = p_num;

filter1(14) = xp1_v; %x variance
filter1(15) = yp1_v; %y variance
filter1(16) = zp1_v; %z variance
filter1(17) = Vp1_v; %V variance
filter1(18) = thetap1_v; %theta variance
filter1(19) = phip1_v; %phi variance
filter1(20) = dxp1_v; %dyp variance
filter1(21) = dyp1_v; %dyp variance
filter1(22) = dzp1_v; %dzp variance
filter1(23) = dVp1_v; %dVp variance
filter1(24) = dthetap1_v; %dthetap variance
filter1(25) = dhip1_v; %dhip variance

filter1(26) = m1_v; %Measurement noise covariance

filter1(27) = W_u;
filter1(28) = W_v;
filter1(29) = W_w;

for i=1:sim_run

```

```

[T_G.cent, O_G.cent, O_C.cent, K_G.S, P_G.S, L_G.cent, L_C.cent] =...
    EPF_A.Function.Final(params, target, obs, Kfilter, filter1);

%Convert to degrees

T_G.cent(:, [5,6,11,12])=rad2deg(T_G.cent(:, [5,6,11,12]));
P_G.S(:, [5,6,11,12])=rad2deg(P_G.S(:, [5,6,11,12]));
L_G.cent(:, [8,9])=rad2deg(L_G.cent(:, [8,9]));
K_G.S(:, [5,6])=rad2deg(K_G.S(:, [5,6]));

sim_T_G.cent(:, :, i)=T_G.cent;
sim_O_G.cent(:, :, i)=O_G.cent;
sim_K_G.S(:, :, i)=K_G.S;
sim_P_G.S(:, :, i)=P_G.S;
sim_L_G.cent(:, :, i)=L_G.cent;

i

end

switch sim_plot
case 1
    fig=1;
    hfig=figure(fig);
    set(hfig, 'Position', [0, 0, 800, 800]); %[x y width height]
    axis equal
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    for i2=4:params(1)
        hold on;
        scatter3(T_G.cent(i2,1), T_G.cent(i2,2), T_G.cent(i2,3), 'b');
        scatter3(O_G.cent(i2,4), O_G.cent(i2,5), O_G.cent(i2,6), 'k');
        scatter3(L_G.cent(i2,1), L_G.cent(i2,2), L_G.cent(i2,3), 'g');
        scatter3(K_G.S(i2,1), K_G.S(i2,2), K_G.S(i2,3), 'm');
        scatter3(P_G.S(i2,1), P_G.S(i2,2), P_G.S(i2,3), 'r');
    end
end

```

```

end

legend('Target','Observation','SLMA','Kalman','EPF-A',...
        'Location','East')

fig=fig+1;

%Comparison of States
time=zeros(params(1),1);
for i2=3:params(1)
    time(i2,1)=i2;
end

%Positions
hfig=figure(fig);
set(hfig,'Position',[0, 0, 800, 1200]); %[x y width height]
a=subplot(3,1,1);
plot([time(1),NaN],[T_G_cent(1,1),NaN],'*-b',...
      [time(1),NaN],[O_G_cent(1,1),NaN],'*-g',...
      [time(1),NaN],[L_G_cent(1,1),NaN],'*-c',...
      [time(1),NaN],[K_G_S(1,1),NaN],'*-m',...
      [time(1),NaN],[P_G_S(1,1),NaN],'*-r');
legend('Target','Measurement','SLMA','EKF-A','EPF-A',...
        'Location','North','Orientation','Horizontal')
set(legend,'Orientation','horizontal','Position',...
      [0.3 .04 .4 .025]);
hold on;
plot(time,T_G_cent(:,1),'b',...
      time,O_G_cent(:,4),'g',...
      time,L_G_cent(:,1),'c',...
      time,K_G_S(:,1),'m',...
      time,P_G_S(:,1),'r');
hold on;
plot(time(1:20:end),T_G_cent((1:20:end),1),'*b',...

```

```

        time(4:20:end), O_G_cent((4:20:end), 4), '*g', ...
        time(8:20:end), L_G_cent((8:20:end), 1), '*c', ...
        time(12:20:end), K_G_S((12:20:end), 1), '*m', ...
        time((16:20:end)), P_G_S((16:20:end), 1), '*r');
title(a, 'x Position');
xlabel(a, 'Time Step');
ylabel(a, 'Position');

b=subplot(3,1,2);
plot(time, T_G_cent(:, 2), 'b', ...
      time, O_G_cent(:, 5), 'g', ...
      time, L_G_cent(:, 2), 'c', ...
      time, K_G_S(:, 2), 'm', ...
      time, P_G_S(:, 2), 'r');
hold on;
plot(time(1:20:end), T_G_cent((1:20:end), 2), '*b', ...
      time(4:20:end), O_G_cent((4:20:end), 5), '*g', ...
      time(8:20:end), L_G_cent((8:20:end), 2), '*c', ...
      time(12:20:end), K_G_S((12:20:end), 2), '*m', ...
      time((16:20:end)), P_G_S((16:20:end), 2), '*r');
title(b, 'y Position');
xlabel(b, 'Time Step');
ylabel(b, 'Position');

c=subplot(3,1,3);
plot(time, T_G_cent(:, 3), 'b', ...
      time, O_G_cent(:, 6), 'g', ...
      time, L_G_cent(:, 3), 'c', ...
      time, K_G_S(:, 3), 'm', ...
      time, P_G_S(:, 3), 'r');
hold on;
plot(time(1:20:end), T_G_cent((1:20:end), 3), '*b', ...

```

```

        time(4:20:end), O_G_cent((4:20:end), 6), '*g', ...
        time(8:20:end), L_G_cent((8:20:end), 3), '*c', ...
        time(12:20:end), K_G_S((12:20:end), 3), '*m', ...
        time((16:20:end)), P_G_S((16:20:end), 3), '*r');
title(c, 'z Position');
xlabel(c, 'Time Step');
ylabel(c, 'Position');
fig=fig+1;

%Velocity Magintude and headings
hfig=figure(fig);
set(hfig, 'Position', [0, 0, 800, 1200]); %[x y width height]
a=subplot(3,1,1);
plot([time(1), NaN], [T_G_cent(1,1), NaN], '*-b', ...
      [time(1), NaN], [L_G_cent(1,1), NaN], '*-c', ...
      [time(1), NaN], [K_G_S(1,1), NaN], '*-m', ...
      [time(1), NaN], [P_G_S(1,1), NaN], '*-r');
legend('Target', 'SLMA', 'EKF-A', 'EPF-A', 'Location', 'North', ...
       'Orientation', 'Horizontal')
set(legend, 'Orientation', 'horizontal', 'Position', ...
      [0.3 .04 .4 .025]);
hold on;
plot(time, T_G_cent(:,4), 'b', ...
      time, L_G_cent(:,7), 'c', ...
      time, K_G_S(:,4), 'm', ...
      time, P_G_S(:,4), 'r');
hold on;
plot(time(1:16:end), T_G_cent((1:16:end), 4), '*b', ...
      time(4:16:end), L_G_cent((4:16:end), 7), '*c', ...
      time(8:16:end), K_G_S((8:16:end), 4), '*m', ...
      time((12:16:end)), P_G_S((12:16:end), 4), '*r');
title(a, 'Velocity Magnitude');

```

```

xlabel(a, 'Time Step');
ylabel(a, 'Velocity');

b=subplot(3,1,2);
plot(time,T_G_cent(:,5), 'b', ...
      time,L_G_cent(:,8), 'c', ...
      time,K_G_S(:,5), 'm', ...
      time,P_G_S(:,5), 'r');
hold on;
plot(time(1:16:end),T_G_cent((1:16:end),5), '*b', ...
      time(4:16:end),L_G_cent((4:16:end),8), '*c', ...
      time(8:16:end),K_G_S((8:16:end),5), '*m', ...
      time((12:16:end)),P_G_S((12:16:end),5), '*r');
title(b, 'Theta Heading');
xlabel(b, 'Time Step');
ylabel(b, 'Heading (deg)');

c=subplot(3,1,3);
plot(time,T_G_cent(:,6), 'b', ...
      time,L_G_cent(:,9), 'c', ...
      time,K_G_S(:,6), 'm', ...
      time,P_G_S(:,6), 'r');
hold on;
plot(time(1:16:end),T_G_cent((1:16:end),6), '*b', ...
      time(4:16:end),L_G_cent((4:16:end),9), '*c', ...
      time(8:16:end),K_G_S((8:16:end),6), '*m', ...
      time((12:16:end)),P_G_S((12:16:end),6), '*r');
title(c, 'Phi Heading');
xlabel(c, 'Time Step');
ylabel(c, 'Heading (deg)');
fig=fig+1;

```

```

%% Mean Errors
for i=2:T
    Error_SLMA(i,:)=abs(L_G_cent(i,[1,2,3,7,8,9])-...
        T_G_cent(i,[1,2,3,4,5,6]));
    Error_Kalman(i,:)=abs(K_G_S(i,:)-T_G_cent(i,[1,2,3,4,5,6]));
    Error_Particle(i,:)=abs(P_G_S(i,[1,2,3,4,5,6])-...
        T_G_cent(i,[1,2,3,4,5,6]));
end

%Return Mean Errors
SLMA=mean(Error_SLMA(T-25:T,:))
Kalman=mean(Error_Kalman(T-25:T,:))
Particle=mean(Error_Particle(T-25:T,:))

otherwise
    message='WILL NOT PLOT, MULTIPLE SIMULATION RUNS'

%% Calculate performace abilities
%NOTE: Although distance and V errors are absolute, heading errors
%can only be up to 180 degrees off for both phi and theta (ie. you
%are pointinig in the complete opposite direction. Theta is
%already between 0 and 180, so aboslute errors may be taken. Phi
%though must be adjusted so that all errors are between 0 and 180
%(ex. an 'error' of 350 is really only an error of 10
%'Best' Values for filter and comparison
range_index=solution_range*sim_run;
for i=2:params(1)
    for i2=1:sim_run
        %Create variable vectors
        t_dist=sqrt(sim_T_G_cent(i,1,i2)^2+...
            sim_T_G_cent(i,2,i2)^2+sim_T_G_cent(i,3,i2)^2);
        k_dist=sqrt(sim_K_G_S(i,1,i2)^2+sim_K_G_S(i,2,i2)^2+...

```

```

        sim_K_G_S(i,3,i2)^2);
f_dist=sqrt(sim_P_G_S(i,1,i2)^2+sim_P_G_S(i,2,i2)^2+...
        sim_P_G_S(i,3,i2)^2);
c_dist=sqrt(sim_L_G_cent(i,1,i2)^2+...
        sim_L_G_cent(i,2,i2)^2+sim_L_G_cent(i,3,i2)^2);

vec_sim_K_G_S(i2,1)=abs(k_dist-t_dist);
vec_sim_K_G_S(i2,[2,3,4])=abs(sim_K_G_S(i,[4,5,6],i2)-...
        sim_T_G_cent(i,[4,5,6],i2));
vec_sim_P_G_S(i2,1)=abs(f_dist-t_dist);
vec_sim_P_G_S(i2,[2,3,4])=abs(sim_P_G_S(i,[4,5,6],i2)-...
        sim_T_G_cent(i,[4,5,6],i2));
vec_sim_L_G_cent(i2,1)=abs(c_dist-t_dist);
vec_sim_L_G_cent(i2,[2,3,4])=...
        abs(sim_L_G_cent(i,[7,8,9],i2)-...
        sim_T_G_cent(i,[4,5,6],i2));

%Phi adjustment
if vec_sim_K_G_S(i2,4)>180
    vec_sim_K_G_S(i2,4)=360-vec_sim_K_G_S(i2,4);
end

if vec_sim_P_G_S(i2,4)>180
    vec_sim_P_G_S(i2,4)=360-vec_sim_P_G_S(i2,4);
end

if vec_sim_L_G_cent(i2,4)>180
    vec_sim_L_G_cent(i2,4)=360-vec_sim_L_G_cent(i2,4);
end
end

svec_sim_K_G_S=sort(vec_sim_K_G_S);

```

```

svec_sim_P_G_S=sort(vec_sim_P_G_S);
svec_sim_L_G_cent=sort(vec_sim_L_G_cent);
error_t_k(i,:)=svec_sim_K_G_S(range_index,:);
error_t_f(i,:)=svec_sim_P_G_S(range_index,:);
error_t_c(i,:)=svec_sim_L_G_cent(range_index,:);
end

temp_error_k=mean(error_t_k(params(1)-50:params(1),:));
temp_error_f=mean(error_t_f(params(1)-50:params(1),:));

%Mean or average error values for filter and comparison
for i=2:params(1)
    for i2=1:sim_run
        %Create variable vectors
        t_dist=sqrt(sim_T_G_cent(i,1,i2)^2+...
            sim_T_G_cent(i,2,i2)^2+sim_T_G_cent(i,3,i2)^2);
        k_dist=sqrt(sim_K_G_S(i,1,i2)^2+sim_K_G_S(i,2,i2)^2+...
            sim_K_G_S(i,3,i2)^2);
        f_dist=sqrt(sim_P_G_S(i,1,i2)^2+sim_P_G_S(i,2,i2)^2+...
            sim_P_G_S(i,3,i2)^2);
        c_dist=sqrt(sim_L_G_cent(i,1,i2)^2+...
            sim_L_G_cent(i,2,i2)^2+sim_L_G_cent(i,3,i2)^2);

        vec_sim_K_G_S(i2,1)=abs(k_dist-t_dist);
        vec_sim_K_G_S(i2,[2,3,4])=abs(sim_K_G_S(i,[4,5,6],i2)-...
            sim_T_G_cent(i,[4,5,6],i2));
        vec_sim_P_G_S(i2,1)=abs(f_dist-t_dist);
        vec_sim_P_G_S(i2,[2,3,4])=abs(sim_P_G_S(i,[4,5,6],i2)-...
            sim_T_G_cent(i,[4,5,6],i2));
        vec_sim_L_G_cent(i2,1)=abs(c_dist-t_dist);
        vec_sim_L_G_cent(i2,[2,3,4])=...

```

```

        abs(sim_L_G_cent(i,[7,8,9],i2)-...
        sim_T_G_cent(i,[4,5,6],i2));

    if vec_sim_K_G_S(i2,4)>180
        vec_sim_K_G_S(i2,4)=360-vec_sim_K_G_S(i2,4);
    end

    if vec_sim_P_G_S(i2,4)>180
        vec_sim_P_G_S(i2,4)=360-vec_sim_P_G_S(i2,4);
    end

    if vec_sim_L_G_cent(i2,4)>180
        vec_sim_L_G_cent(i2,4)=360-vec_sim_L_G_cent(i2,4);
    end
end

error_m_k(i,1)=mean(vec_sim_K_G_S(:,1));
error_m_k(i,2)=mean(vec_sim_K_G_S(:,2));
error_m_k(i,3)=mean(vec_sim_K_G_S(:,3));
error_m_k(i,4)=mean(vec_sim_K_G_S(:,4));

error_m_f(i,1)=mean(vec_sim_P_G_S(:,1));
error_m_f(i,2)=mean(vec_sim_P_G_S(:,2));
error_m_f(i,3)=mean(vec_sim_P_G_S(:,3));
error_m_f(i,4)=mean(vec_sim_P_G_S(:,4));

error_m_c(i,1)=mean(vec_sim_L_G_cent(:,1));
error_m_c(i,2)=mean(vec_sim_L_G_cent(:,2));
error_m_c(i,3)=mean(vec_sim_L_G_cent(:,3));
error_m_c(i,4)=mean(vec_sim_L_G_cent(:,4));
end

```

```

%Bias (Overall mean for certain range versus simulation number)
bias_start=params(1)-50;
bias_stop=params(1);
sim_runx=5;
for i=1:sim_run
    for i2=bias_start:bias_stop
        t_dist=sqrt(sim_T_G_cent(i2,1,i)^2+...
            sim_T_G_cent(i2,2,i)^2+sim_T_G_cent(i2,2,i)^2);
        k_dist=sqrt(sim_K_G_S(i2,1,i)^2+sim_K_G_S(i2,2,i)^2+...
            sim_K_G_S(i2,2,i)^2);
        f_dist=sqrt(sim_P_G_S(i2,1,i)^2+sim_P_G_S(i2,2,i)^2+...
            sim_P_G_S(i2,2,i)^2);
        c_dist=sqrt(sim_L_G_cent(i2,1,i)^2+...
            sim_L_G_cent(i2,2,i)^2+sim_L_G_cent(i2,2,i)^2);

        vec2_sim_K_G_S(i2,1)=k_dist-t_dist;
        vec2_sim_K_G_S(i2,[2,3,4])=sim_K_G_S(i2,[4,5,6],i)-...
            sim_T_G_cent(i2,[4,5,6],i);
        vec2_sim_P_G_S(i2,1)=f_dist-t_dist;
        vec2_sim_P_G_S(i2,[2,3,4])=sim_P_G_S(i2,[4,5,6],i)-...
            sim_T_G_cent(i2,[4,5,6],i);
        vec2_sim_L_G_cent(i2,1)=c_dist-t_dist;
        vec2_sim_L_G_cent(i2,[2,3,4])=...
            sim_L_G_cent(i2,[7,8,9],i)-sim_T_G_cent(i2,[4,5,6],i);

        if vec2_sim_K_G_S(i2,4)>180
            vec2_sim_K_G_S(i2,4)=360-vec2_sim_K_G_S(i2,4);
        end

        if vec2_sim_P_G_S(i2,4)>180
            vec2_sim_P_G_S(i2,4)=360-vec2_sim_P_G_S(i2,4);
        end
    end
end

```

```

    if vec2_sim_LG_cent(i2,4)>180
        vec2_sim_LG_cent(i2,4)=360-vec2_sim_LG_cent(i2,4);
    end

    if vec2_sim_KG_S(i2,4)<-180
        vec2_sim_KG_S(i2,4)=abs(vec2_sim_KG_S(i2,4))-360;
    end

    if vec2_sim_PG_S(i2,4)<-180
        vec2_sim_PG_S(i2,4)=abs(vec2_sim_PG_S(i2,4))-360;
    end

    if vec2_sim_LG_cent(i2,4)<-180
        vec2_sim_LG_cent(i2,4)=...
            abs(vec2_sim_LG_cent(i2,4))-360;
    end
end

temp_bias_k(i,1)=mean(vec2_sim_KG_S(:,1),1);
temp_bias_k(i,2)=mean(vec2_sim_KG_S(:,2),1);
temp_bias_k(i,3)=mean(vec2_sim_KG_S(:,3),1);
temp_bias_k(i,4)=mean(vec2_sim_KG_S(:,4),1);

temp_bias_p(i,1)=mean(vec2_sim_PG_S(:,1),1);
temp_bias_p(i,2)=mean(vec2_sim_PG_S(:,2),1);
temp_bias_p(i,3)=mean(vec2_sim_PG_S(:,3),1);
temp_bias_p(i,4)=mean(vec2_sim_PG_S(:,4),1);

temp_bias_c(i,1)=mean(vec2_sim_LG_cent(:,1),1);
temp_bias_c(i,2)=mean(vec2_sim_LG_cent(:,2),1);
temp_bias_c(i,3)=mean(vec2_sim_LG_cent(:,3),1);

```

```

        temp_bias_c(i,4)=mean(vec2_sim_LG_cent(:,4),1);
        count=i
    end

    time=zeros(params(1),1);
    for i2=2:params(1)
        time(i2,1)=i2;
    end

    fig=1;
    %TER
    hfig=figure('name','Threshold Error');
    set(hfig,'Position',[0, 0, 800, 1200]); %[x y width height]

    a=subplot(4,1,1);
    plot(time,error_t_c(:,1),'g',...
        time,error_t_k(:,1),'m',...
        time,error_t_f(:,1),'r');
    legend('Comparison','Kalman','EPF-A','Location','NorthEast')
    title(a,'Distance Error');
    xlabel(a,'Time Step');
    ylabel(a,'Error');

    b=subplot(4,1,2);
    plot(time,error_t_c(:,2),'g',...
        time,error_t_k(:,2),'m',...
        time,error_t_f(:,2),'r',...
        time,T_G_cent(:,4),'b');
    legend('Comparison','Kalman','EPF-A','Target',...
        'Location','NorthEast')
    title(b,'Velocity Error');
    xlabel(b,'Time Step');
    ylabel(b,'Error');

```

```

c=subplot(4,1,3);
plot(time,error_t_c(:,3),'g',...
      time,error_t_k(:,3),'m',...
      time,error_t_f(:,3),'r',...
      time,T_G_cent(:,5),'b');
legend('Comparsion','Kalman','EPF-A','Target',...
       'Location','NorthEast')
title(c,'Theta Heading Error');
xlabel(c,'Time Step');
ylabel(c,'Error (Deg)');

d=subplot(4,1,4);
plot(time,error_t_c(:,4),'g',...
      time,error_t_k(:,4),'m',...
      time,error_t_f(:,4),'r',...
      time,T_G_cent(:,6),'b');
legend('Comparison','Kalman','EPF-A','Target',...
       'Location','NorthEast')
title(d,'Phi Heading Error');
xlabel(d,'Time Step');
ylabel(d,'Error (Deg)');
fig=fig+1;

%MAE
hfig=figure('name','Mean Error');
set(hfig,'Position',[0, 0, 800, 1200]); %[x y width height]
a=subplot(4,1,1);
plot(time,error_m_c(:,1),'g',...
      time,error_m_k(:,1),'m',...
      time,error_m_f(:,1),'r');
legend('Comparison','Kalman','EPF-A','Location','NorthEast')
title(a,'Distance Error');

```

```

xlabel(a, 'Time Step');
ylabel(a, 'Error');

b=subplot(4,1,2);
plot(time,error_m_c(:,2), 'g', ...
      time,error_m_k(:,2), 'm', ...
      time,error_m_f(:,2), 'r', ...
      time,T_G_cent(:,4), 'b');
legend('Comparison', 'Kalman', 'EPF-A', 'Target', ...
       'Location', 'NorthEast')
title(b, 'Velocity Error');
xlabel(b, 'Time Step');
ylabel(b, 'Error');

c=subplot(4,1,3);
plot(time,error_m_c(:,3), 'g', ...
      time,error_m_k(:,3), 'm', ...
      time,error_m_f(:,3), 'r', ...
      time,T_G_cent(:,5), 'b');
legend('Comparison', 'Kalman', 'EPF-A', 'Target', ...
       'Location', 'NorthEast')
title(c, 'Theta Heading Error');
xlabel(c, 'Time Step');
ylabel(c, 'Error (Deg)');

d=subplot(4,1,4);
plot(time,error_m_c(:,4), 'g', ...
      time,error_m_k(:,4), 'm', ...
      time,error_m_f(:,4), 'r', ...
      time,T_G_cent(:,6), 'b');
legend('Comparison', 'Kalman', 'EPF-A', 'Target', ...
       'Location', 'NorthEast')

```

```

        title(d, 'Phi Heading Error');
        xlabel(d, 'Time Step');
        ylabel(d, 'Error (Deg)');
        fig=fig+1;

        %Final 50 step metrics
        best_error_c=mean(error_t_c(params(1)-50:params(1),:))
        best_error_k=mean(error_t_k(params(1)-50:params(1),:))
        best_error_f=mean(error_t_f(params(1)-50:params(1),:))

        %Final 50 step metrics
        mean_error_c=mean(error_m_c(params(1)-50:params(1),:))
        mean_error_k=mean(error_m_k(params(1)-50:params(1),:))
        mean_error_f=mean(error_m_f(params(1)-50:params(1),:))
end

```

A.4.2 EPF A Function Final.

```

function [T_G_cent, O_G_cent, O_C_cent, K_G_cent, P_G_S, L_G_cent,...
        L_C_cent] = Particle_Filter_A_Function_1(params, target, obs,...
        Kfilter, filter1)

%% This function is Particle Filter A, allowing for multiple scenarios or
%% simulation runs. Inputs are contained within a separate .mfile that
%% executes this file.

%% Initial conditions and setup
rng('shuffle') %Shuffle random numbers

%System parameters
T    =    params(1); %Number of iterations
dt   =    params(2); %Time step

```

```

% Rotation matrix between global and camera frames
Rot_y=@(a) [cos(a), 0, -sin(a); 0,1,0; sin(a), 0, cos(a)];
Rot_x=@(a) [1,0,0;0,cos(a), -sin(a); 0,sin(a), cos(a)];
Rot_z=@(a) [cos(a),-sin(a),0;sin(a), cos(a),0;0,0,1];
PlotDCM=@(A,O) plot3(cumsum([O(1),A(1,1)]),cumsum([O(2),A(2,1)]),...
    cumsum([O(3),A(3,1)]),'r-',...
    cumsum([O(1),A(1,2)]),cumsum([O(2),A(2,2)]),cumsum([O(3),A(3,2)]),'g-',...
    cumsum([O(1),A(1,3)]),cumsum([O(2),A(2,3)]),cumsum([O(3),A(3,3)]),'b-',...
    'linewidth',5);

%% Target parameters
%Truth conditions, the initial conditions of the centroid
x = target(1);
y = target(2);
z = target(3);
V = target(4);
theta = deg2rad(target(5));
phi = deg2rad(target(6));
dx = target(7);
dy = target(8);
dz = target(9);
dV = target(10);
dtheta = deg2rad(target(11));
dphi = deg2rad(target(12));
xdot = target(13);
ydot = target(14);
zdot = target(15);

T_G_cent(1,:)=[x y z V theta phi dx dy dz dV dtheta dphi xdot ydot zdot];

%Process Noise
TS_V(1)=target(16);

```

```

TS_V(2)=target(17);
TS_V(3)=target(18);

%Preallocation
T_C_cent=zeros(T,12);
T_G_Rotdata=zeros(T,3);
T_RotG2C=zeros(3,3,T);

%% Measurement Parameters
%Initial point locations (for a cube), based of T centroid
O_G_point_a=[T_G_cent(1)-.5 T_G_cent(2)-.5 T_G_cent(3)-.5];
O_G_point_b=[T_G_cent(1)+.5 T_G_cent(2)-.5 T_G_cent(3)-.5];
O_G_point_c=[T_G_cent(1)-.5 T_G_cent(2)+.5 T_G_cent(3)-.5];
O_G_point_d=[T_G_cent(1)-.5 T_G_cent(2)-.5 T_G_cent(3)+.5];
O_G_point_e=[T_G_cent(1)-.5 T_G_cent(2)+.5 T_G_cent(3)+.5];
O_G_point_f=[T_G_cent(1)+.5 T_G_cent(2)-.5 T_G_cent(3)+.5];
O_G_point_g=[T_G_cent(1)+.5 T_G_cent(2)+.5 T_G_cent(3)-.5];
O_G_point_h=[T_G_cent(1)+.5 T_G_cent(2)+.5 T_G_cent(3)+.5];

%Form the initial cube based of points
O_G_points=[O_G_point_a;O_G_point_b;O_G_point_f;O_G_point_h;...
    O_G_point_g;O_G_point_c;O_G_point_a;O_G_point_d;...
    O_G_point_e;O_G_point_h;O_G_point_f;O_G_point_d;...
    O_G_point_e;O_G_point_c;O_G_point_g;O_G_point_b];

%Number of points
O_num = 16;

%Preallocation
O_G_cent=zeros(T,6);
O_C_cent=zeros(T,3);
O_RotG2C=zeros(3,3,T);

```

```

O_C_points=zeros(O_num,3,T);

%Measurement noise covariances
O_V_M(1)      =   obs(1);
O_V_M(2)      =   obs(2);
O_V_M(3)      =   obs(3);

%Match initial conditions
O_G_cent(1,:)= [T_G_cent(1,[1,2,3]), T_G_cent(1,[1,2,3])];

%% Kalman Parameters
%Control law
K_u=[Kfilter(7); Kfilter(8); Kfilter(9)];
%State transition model matrix
K_A=[1 0 0 dt 0 0;...
     0 1 0 0 dt 0;...
     0 0 1 0 0 dt;...
     0 0 0 1 0 0;...
     0 0 0 0 1 0;...
     0 0 0 0 0 1];
%Control mode input
K_B=[dt^2/2 0 0;...
     0 dt^2/2 0;...
     0 0 dt^2/2;...
     dt 0 0;...
     0 dt 0;...
     0 0 dt];
%Observation model
K_C=[1 0 0 0 0 0;...
     0 1 0 0 0 0;...
     0 0 1 0 0 0];
%Measurement noise variance values

```

```

K_VM=[Kfilter(16) Kfilter(17) Kfilter(18)];

%Process Noise Covariance
K_Ex=[Kfilter(10) 0 0 0 0 0;...
      0 Kfilter(11) 0 0 0 0;...
      0 0 Kfilter(12) 0 0 0;...
      0 0 0 Kfilter(13) 0 0;...
      0 0 0 0 Kfilter(14) 0;...
      0 0 0 0 0 Kfilter(15)];

%Measurement noise variance matrix
K_Em=[Kfilter(16) 0 0;...
      0 Kfilter(17) 0;...
      0 0 Kfilter(18)];

%Estimate of initial target position variance matrix
K_P=K_Ex;

%Initial states
xk1=Kfilter(1); yk1=Kfilter(2); zk1=Kfilter(3);
Vk1=Kfilter(4); thetak1=Kfilter(5); phik1=Kfilter(6);

%Linearize initial conditions
KS(1,:)=[xk1 yk1 zk1...
          Vk1*sind(thetak1)*cosd(phik1)*dt...
          Vk1*cosd(thetak1)*dt...
          Vk1*sind(thetak1)*sind(phik1)*dt];

%% Particle A Parameters (U,V,W)
%Initial conditions
xp1      = filter1(1);
yp1      = filter1(2);
zp1      = filter1(3);
Vp1      = filter1(4);
thetap1  = filter1(5);
phip1    = filter1(6);

```

```

dxp1    =    filter1(7);
dyp1    =    filter1(8);
dzp1    =    filter1(9);
dVp1     =    filter1(10);
dthetap1 =    filter1(11);
dhip1    =    filter1(12);

xdotp1  =    Vp1*sin(thetap1)*cos(hip1)*dt;
ydotp1  =    Vp1*cos(thetap1)*dt;
zdotp1  =    Vp1*sin(thetap1)*sin(hip1)*dt;

P_G_S(1,:)=[xp1 yp1 zp1 Vp1 thetap1 hip1 dxp1 dyp1 dzp1 dVp1 dthetap1...
            dhip1 xdotp1 ydotp1 zdotp1];

%Number of particles
P_num    =    filter1(13);

%Variances
P_V_S(1)  =    filter1(14); %x variance
P_V_S(2)  =    filter1(15); %y variance
P_V_S(3)  =    filter1(16); %z variance
P_V_S(4)  =    filter1(17); %V variance
P_V_S(5)  =    filter1(18); %theta variance
P_V_S(6)  =    filter1(19); %phi varaince

P_V_S(7)  =    filter1(20); %dyp
P_V_S(8)  =    filter1(21); %dyp
P_V_S(9)  =    filter1(22); %dzp
P_V_S(10) =    filter1(23); %dVp
P_V_S(11) =    filter1(24); %dthetap
P_V_S(12) =    filter1(25); %dhip

```

```

P_V_M    =    filter1(26); %Measurement noise covariance

%Weighing matrix
W_u      =    filter1(27); %u observation weight
W_v      =    filter1(28); %v observation weight
W_w      =    filter1(29); %w observation weight

P_W = [W_u 0 0; 0 W_v 0; 0 0 W_w];

%Initial particles for global state
P_G_S_part=zeros(P_num,15,T); %Preallocation of particle matrix
for i2=1:P_num
    P_G_S_part(i2,:,1)=[P_G_S(1,1)+sqrt(P_V_S(1))*randn;...
        P_G_S(1,2)+sqrt(P_V_S(2))*randn;...
        P_G_S(1,3)+sqrt(P_V_S(3))*randn;...
        P_G_S(1,4)+sqrt(P_V_S(4))*randn;...
        P_G_S(1,5)+sqrt(P_V_S(5))*randn;...
        P_G_S(1,6)+sqrt(P_V_S(6))*randn;...
        P_G_S(1,7)+sqrt(P_V_S(7))*randn;...
        P_G_S(1,8)+sqrt(P_V_S(8))*randn;...
        P_G_S(1,9)+sqrt(P_V_S(9))*randn;...
        P_G_S(1,10)+sqrt(P_V_S(10))*randn;...
        P_G_S(1,11)+sqrt(P_V_S(11))*randn;...
        P_G_S(1,12)+sqrt(P_V_S(12))*randn;...
        P_G_S_part(i2,10,1);...
        P_G_S_part(i2,11,1);...
        P_G_S_part(i2,12,1)];
end

%Preallocation
P_G_S_part_u=zeros(P_num,15,T);

```

```

P_RotG2C=zeros(3,3,T);
P_C_O_part=zeros(P_num,3,T);
P_P_O_diff=zeros(P_num,3,T); %NEED TO CHANGE BACK TO 5 if not W
P_P_W=zeros(T,P_num);

%% SLMA
L_G_cent=zeros(T,9);
L_C_cent(1,[1,2,3])=O_C_cent(1,[1,2,3]);
L_C_cent(1,[4,5,6])=[0,0,0];

for i=2:T %i=1 is initial conditions
%% Truth Centroid
    %Update truth centroid position
    %Position values will NOT be future values: the x used here is based
    %off of the previous x plus the previous delta_x. Thus, delta and dot
    %values pertain to the x in the same time step NOT a future x

    x=T_G_cent(i-1,1)+T_G_cent(i-1,7);
    y=T_G_cent(i-1,2)+T_G_cent(i-1,8);
    z=T_G_cent(i-1,3)+T_G_cent(i-1,9);
    V=T_G_cent(i-1,4)+T_G_cent(i-1,10);
    theta=T_G_cent(i-1,5)+T_G_cent(i-1,11);
    phi=T_G_cent(i-1,6)+T_G_cent(i-1,12);

    xdot=V*sin(theta)*cos(phi)*dt;
    ydot=V*cos(theta)*dt;
    zdot=V*sin(theta)*sin(phi)*dt;

    V_dot=T_G_cent(i-1,10)/dt;
    theta_dot=T_G_cent(i-1,11)/dt;
    phi_dot=T_G_cent(i-1,12)/dt;

    dx=V*sin(theta)*cos(phi)*dt+...

```

```

V_dot*sin(theta)*cos(phi)*(dt^2/2)+...
theta_dot*V*cos(theta)*cos(phi)*(dt^2/2)+...
-phi_dot*V*sin(theta)*sin(phi)*(dt^2/2);

dy=V*cos(theta)*dt+...
V_dot*cos(theta)*(dt^2/2)-...
theta_dot*V*sin(theta)*(dt^2/2);

dz=V*sin(theta)*sin(phi)*dt+...
V_dot*sin(theta)*sin(phi)*(dt^2/2)+...
theta_dot*V*cos(theta)*sin(phi)*(dt^2/2)+...
phi_dot*V*sin(theta)*cos(phi)*(dt^2/2);

dV=V_dot*dt;
dtheta=theta_dot*dt;
dphi=phi_dot*dt;

%Add Process Noise
V=V+sqrt(TS_V(1))*randn;
theta=theta+sqrt(TS_V(2))*randn;
phi=phi+sqrt(TS_V(3))*randn;

%New centroid positions
T_G_cent(i,1)=x;
T_G_cent(i,2)=y;
T_G_cent(i,3)=z;
T_G_cent(i,4)=V;
T_G_cent(i,5)=theta;
T_G_cent(i,6)=phi;
T_G_cent(i,7)=dx;
T_G_cent(i,8)=dy;
T_G_cent(i,9)=dz;

```

```

T_G_cent(i,10)=dV;
T_G_cent(i,11)=dtheta;
T_G_cent(i,12)=dphi;
T_G_cent(i,13)=xdot;
T_G_cent(i,14)=ydot;
T_G_cent(i,15)=zdot;

%Rotate to camera orientation
%States in the camera orientation are u,v,w and dots
%Generate angles and rotation DCM
%Camera angles assume the camera tracks the truth centroid perfectly
T_G_Rotdata(i,1)=sqrt(T_G_cent(i,1)^2+T_G_cent(i,2)^2+T_G_cent(i,3)^2);
%Pan (phi)
T_G_Rotdata(i,2)=atan2(T_G_cent(i,3),T_G_cent(i,1));
%Tilt, note: tilts from y axis (positive down), so 0 is on y-axis
%(theta)
T_G_Rotdata(i,3)=acos(T_G_cent(i,2)/T_G_Rotdata(i,1));

%Calculate rate of rotation (phi dot), x
T_G_Rotdata(i,4)=(T_G_Rotdata(i,2)-T_G_Rotdata(i-1,2))/dt;

%Calculate rate of tilt (theta dot), y
T_G_Rotdata(i,5)=(T_G_Rotdata(i,3)-T_G_Rotdata(i-1,3))/dt;

%Create DCM based on current rotation angles
T_DCM_G2C=@(angz) Rot_y(angz(1)-pi/2)*Rot_x(angz(2)-pi/2)*Rot_z(pi/2);
T_RotG2C(:, :, i)=T_DCM_G2C(T_G_Rotdata(i, [2,3])).';

%Rotate centroid to camera frame
T_C_cent(i, [1,2,3])=T_RotG2C(:, :, i)*T_G_cent(i, [1,2,3]); %u,v,w
T_C_cent(i, [4,5,6])=T_RotG2C(:, :, i)*T_G_cent(i, [13,14,15]); %dot u v w

```

```

%% Observation Centroid and Points
%Update global point position (CUBE)
    for i2=1:O_num
        O_G_points(i2,1,i)=O_G_points(i2,1,i-1)+T_G_cent(i-1,7);
        O_G_points(i2,2,i)=O_G_points(i2,2,i-1)+T_G_cent(i-1,8);
        O_G_points(i2,3,i)=O_G_points(i2,3,i-1)+T_G_cent(i-1,9);
    end

%Calculate observed centroid based on mean of points
    O_G_cent(i,1)=mean(O_G_points(:,1,i));
    O_G_cent(i,2)=mean(O_G_points(:,2,i));
    O_G_cent(i,3)=mean(O_G_points(:,3,i));

%Rotate to camera orientation (use same cam rot angles as in truth)
    %Rotate points to camera orientation
    O_RotG2C(:, :, i)=T_RotG2C(:, :, i);
    O_noise=[sqrt(O_V_M(1))*randn; sqrt(O_V_M(2))*randn;...
            sqrt(O_V_M(3))*randn];
    for i2=1:O_num
        O_C_points(i2, :, i)=O_RotG2C(:, :, i)*O_G_points(i2, :, i)'+O_noise;
    end

%Calculate observed centroid (in camera frame)
    O_C_cent(i,1)=mean(O_C_points(:,1,i));
    O_C_cent(i,2)=mean(O_C_points(:,2,i));
    O_C_cent(i,3)=mean(O_C_points(:,3,i));

    T_RotC2G(:, :, i)=transpose(T_RotG2C(:, :, i));
    O_G_cent(i, [4,5,6])=T_RotC2G(:, :, i)*O_C_cent(i, [1,2,3])';

%% Kalman Filter
KS_t = K_A * KS(i-1, :) + K_B*K_u;

```

```

KS(i,:)=KS_t';

K_P=K_A*K_P*K_A'+K_Ex';

K_K=K_P*K_C'*inv(K_C*K_P*K_C'+K_Em);

%Rotate observations from camera to global
% K_comp(i,[1,2,3])=T_RotC2G(:, :, i)*O_C_cent(i,[1,2,3])';

%Update state estimate
KS_t=KS(i,:)' + K_K*(T_G_cent(i,[1,2,3])'-K_C*KS(i,:)');
KS(i,:)=KS_t';

%Update covariance estimation
K_P=(eye(6)-K_K*K_C)*K_P;

%Convert to global states
K_G_cent(i,1)=KS(i,1);
K_G_cent(i,2)=KS(i,2);
K_G_cent(i,3)=KS(i,3);

% K_G_cent(i,[1,2,3])=KS([1,2,3]);
K_G_cent(i,4)=abs(sqrt(KS(i,1)^2+KS(i,2)^2+KS(i,3)^2));
K_G_cent(i,5)=acos(KS(i,5)/K_G_cent(i,4));
K_G_cent(i,6)=atan2(KS(i,6),KS(i,4));

%% Particle Centroid and Points (U,V,W)
%Update particles (state and observations)
%As with the Truth cent, take in the previous time step values
for i2=1:P_num
    %Update state
    x=P_G_S_part(i2,1,i-1)+P_G_S_part(i2,7,i-1);
    y=P_G_S_part(i2,2,i-1)+P_G_S_part(i2,8,i-1);

```

```

z=P_G_S_part (i2,3,i-1)+P_G_S_part (i2,9,i-1);

V=P_G_S_part (i2,4,i-1)+P_G_S_part (i2,10,i-1)+sqrt (P_V_S (4)) *randn;
theta=P_G_S_part (i2,5,i-1)+P_G_S_part (i2,11,i-1)+...
    sqrt (P_V_S (5)) *randn;
phi=P_G_S_part (i2,6,i-1)+P_G_S_part (i2,12,i-1)+...
    sqrt (P_V_S (6)) *randn;

%Angle adjustment: ensure V is not negative, angles in proper range
if (V<0)
    V=abs (V);
end

%Fix and reduce angles
theta=wrapTo2Pi (theta);
if (theta>pi ())
    theta_a=theta-pi ();
    theta=pi ()-theta_a;
end

xdot=V*sin (theta)*cos (phi)*dt;
ydot=V*cos (theta)*dt;
zdot=V*sin (theta)*sin (phi)*dt;

V_dot=T_G_cent (i-1,10)/dt;
theta_dot=T_G_cent (i-1,11)/dt;
phi_dot=T_G_cent (i-1,12)/dt;

dx=V*sin (theta)*cos (phi)*dt+...
    V_dot*sin (theta)*cos (phi)*(dt^2/2)+...
    theta_dot*V*cos (theta)*cos (phi)*(dt^2/2)+...
    -phi_dot*V*sin (theta)*sin (phi)*(dt^2/2);

```

```

dy=V*cos(theta)*dt+...
    V_dot*cos(theta)*(dt^2/2)-...
    theta_dot*V*sin(theta)*(dt^2/2);

dz=V*sin(theta)*sin(phi)*dt+...
    V_dot*sin(theta)*sin(phi)*(dt^2/2)+...
    theta_dot*V*cos(theta)*sin(phi)*(dt^2/2)+...
    phi_dot*V*sin(theta)*cos(phi)*(dt^2/2);

dV=V_dot*dt;
dtheta=theta_dot*dt;
dphi=phi_dot*dt;

%New centroid positions
P_G_S_part_u(i2,1,i)=x;
P_G_S_part_u(i2,2,i)=y;
P_G_S_part_u(i2,3,i)=z;
P_G_S_part_u(i2,4,i)=V;
P_G_S_part_u(i2,5,i)=theta;
P_G_S_part_u(i2,6,i)=phi;
P_G_S_part_u(i2,7,i)=dx;
P_G_S_part_u(i2,8,i)=dy;
P_G_S_part_u(i2,9,i)=dz;
P_G_S_part_u(i2,10,i)=dV;
P_G_S_part_u(i2,11,i)=dtheta;
P_G_S_part_u(i2,12,i)=dphi;
P_G_S_part_u(i2,13,i)=xdot;
P_G_S_part_u(i2,14,i)=ydot;
P_G_S_part_u(i2,15,i)=zdot;

```

```

%Observation Update (ie. what we think the camera will see based on

```

```

%the states)

%Rotate to camera frame (use same angles as in Truth)
P_RotG2C(:, :, i) = T_RotG2C(:, :, i);
P_C_O_part(i2, [1, 2, 3], i) = P_RotG2C(:, :, i) * ...
    P_G_S_part_u(i2, [1, 2, 3], i)';

%Calculate difference between observation (measurement) and filter
%prediction
P_P_O_diff(i2, :, i) = O_C_cent(i, :) - P_C_O_part(i2, :, i);

%Weights to be used
P_P_O_rawW(i, i2) = P_P_O_diff(i2, :, i) * P_W * P_P_O_diff(i2, :, i)';

%Weight particles
P_P_W(i, i2) = (1/sqrt(2*pi*P_V_M)) * exp(-(P_P_O_diff(i2, :, i) * P_W * ...
    P_P_O_diff(i2, :, i)') / (2*P_V_M));
end

%Normalize to form a probability distribution (ie. sums to 1)
P_P_W(i, :) = P_P_W(i, :) ./ sum(P_P_W(i, :));

%Resampling: from this new distribution, we randomly resample from it
%to generate new estimate particles
for i2=1:P_num
    P_P_get(i, i2) = find(rand <= cumsum(P_P_W(i, :)), 1);
    P_G_S_part(i2, :, i) = P_G_S_part_u(P_P_get(i, i2), :, i);
end

%The final estimate, state, is a metric of the final resampling
P_G_S(i, :) = mean(P_G_S_part(:, :, i));

%% SLMA

```

```

O_C.cent(1,[1,2,3])=O_G.cent(1,[1,2,3]);

L_C.cent(i,1)    =    O_C.cent(i-1,1)+L_C.cent(i-1,4)*dt;
L_C.cent(i,2)    =    O_C.cent(i-1,2)+L_C.cent(i-1,5)*dt;
L_C.cent(i,3)    =    O_C.cent(i-1,3)+L_C.cent(i-1,6)*dt;

%Velocities
L_C.cent(i,4)    =    (O_C.cent(i,1)-O_C.cent(i-1,1))/dt;
L_C.cent(i,5)    =    (O_C.cent(i,2)-O_C.cent(i-1,2))/dt;
L_C.cent(i,6)    =    (O_C.cent(i,3)-O_C.cent(i-1,3))/dt;

%Rotate to global from camera (inverse of DCM)
L_G.cent(i,[1,2,3])=T_RotC2G(:, :, i)*L_C.cent(i,[1,2,3])';

L_G.cent(i,4)=(L_G.cent(i,1)-L_G.cent(i-1,1))/dt;
L_G.cent(i,5)=(L_G.cent(i,2)-L_G.cent(i-1,2))/dt;
L_G.cent(i,6)=(L_G.cent(i,3)-L_G.cent(i-1,3))/dt;

V    =    sqrt(L_G.cent(i,4)^2+L_G.cent(i,5)^2+L_G.cent(i,6)^2);
theta    =    acos(L_G.cent(i,5)/V);
phi      =    atan2(L_G.cent(i,6),L_G.cent(i,4));

L_G.cent(i,7)    =    V;
L_G.cent(i,8)    =    theta;
L_G.cent(i,9)    =    phi;
end

%Angle adjustment: ensure V is not negative, angles in proper range
for i=2:T
    %Target
    T_G.cent(i,5)=wrapTo2Pi(T_G.cent(i,5));
    T_G.cent(i,6)=wrapTo2Pi(T_G.cent(i,6));

```

```

if(T_G_cent(i,4)<0)
    T_G_cent(i,4)=abs(T_G_cent(i,4));
    T_G_cent(i,6)=wrapTo2Pi(T_G_cent(i,6)+pi());
    T_G_cent(i,5)=pi()-T_G_cent(i,5);
end

%Fix and reduce angles
if(T_G_cent(i,5)>pi())
    theta_a=T_G_cent(i,5)-pi();
    T_G_cent(i,5)=pi()-theta_a;
    T_G_cent(i,6)=wrapTo2Pi(T_G_cent(i,6)+pi());
end

%Kalman
phi=K_G_cent(i,6);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
K_G_cent(i,6)=abs(phi);

%Fix and reduce angles

```

```

theta=K_G_cent(i,5);
theta=wrapTo2Pi(theta);
if(theta>pi())
    theta_a=theta-pi();
    theta=pi()-theta_a;
end
K_G_cent(i,5)=theta;

%EPF-A
%Only need to adjust phi, v is already only absolute and theta has
%already been constrained
%DO NOT USE WRAPTO2PI!!!
phi=P_G_S(i,6);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
P_G_S(i,6)=abs(phi);

%SLMA
%Angle adjustment: ensure V is not negative, angles in proper range
phi=L_G_cent(i,9);
if (phi > 2*pi())

```

```

        mult=floor(phi/(2*pi()));
        phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end

L_G_cent(i,9)=abs(phi);

%Fix and reduce angles
theta=L_G_cent(i,8);
theta=wrapTo2Pi(theta);
if(theta>pi())
    theta_a=theta-pi();
    theta=pi()-theta_a;
end

L_G_cent(i,8)=theta;
end

```

A.5 Evaluated Particle Filter B

Two functions executed and plotted EPF-B, EPF-A, SLMA, and SLMB. The first function, EPF_B_Execute_Final.m, provides the initial conditions to the second function, EPF_B_Function_Final.m, which executes the filters.

A.5.1 *EPF B Execute Final.*

```

%% This script analyzes EPF-B
clear all; close all; clc;

%% Defaults
% Change default axes fonts
set(0,'DefaultAxesFontName','Times New Roman');
set(0,'DefaultAxesFontSize', 12);

% Change default text fonts
set(0,'DefaultTextFontName', 'Times New Roman');
set(0,'DefaultTextFontSize', 12);

%% Parameters
%Number of simulation runs
sim_run=1;
sim_plot=sim_run;
%Threshold Error Range
solution_range = .9;
%Iteration Number
params(1)    =    5;
%Time Steps
params(2)    =    .1;
%Focal length
params(3)    =    2000;

%% Target Parameters
%Straight Line, x-axis: 1
%Straight Line, y-axis: 2
%Straight Line, z-axis: 3
%Straight Line, -x-axis: 4
%Straight Line, -y-axis: 5
%Straight Line, -z-axis: 6
%Circle, x/y, about z: 7

```

```

%Circle, x/z, about y:  8
%Circle, y/z, about x:  9
%Filter B:              10
%MANUAL, see below:     0
target_scenario = 10;

%Process Variance
%0.1:  2
%None:  0
P_variance_scenario = 2;

%No measurment variance:  1
%5 Measurement variance:  2
%10 Measurement variance: 3
%MANUAL:                  0
m_variance_scenario =2;

%% EPF-B/EPF-A Parameters
%With target intial conditions: 1
%Without, at 0 (x,y,z):         2
%For circle, x/y about z:       3
%For circle, x/z about y:       3
%For circle, y/z about y:       3
%MANUAL:                       0
filter1_scenario = 0;
filter2_scenario = 0;

%Number of particles
p_num = 500;
p2_num = 1500;

%Weights EPF-A

```

```

%All equal: 1
%MANUAL:      0
filter1_weight =1;

%Weights EPF-B
%All equal: 1
%MANUAL:      0
filter2_weight = 0;

%Filter variances
%Scenario 1:   1
%MANUAL:       0
filter1_variance = 0;
filter2_variance = 0;

%Measurment Variance
%Match measurment:  1
%Variance of .01:    2
%Variance of .1;     3
%MANUAL:             0
%NOTE: CANNOT BE 0
filter1_measurement_variance = 1;
filter2_measurement_variance = 1;

%Type of correction to use for filter 2
%Noise on vector:    1
%Jacobian:           2
%No correction:      0
correction=0;

%Manual values
%Target

```

```

manual(1) = 0; %x
manual(2) = 0; %y
manual(3) = 5; %z
manual(4) = 1; %V
manual(5) = 90; %theta
manual(6) = 0; %phi
manual(7) = 0; %dx
manual(8) = 0; %dy
manual(9) = 0; %dz
manual(10) = 0; %dV
manual(11) = 0; %dtheta
manual(12) = 0; %dphi
manual(13) = 0; %x_dot
manual(14) = 0; %y_dot
manual(15) = 0; %z_dot

manual(16) = 0; %u measurement variance
manual(17) = 0; %v measurement variance
manual(18) = 0; %w measurement variance

%EPF-A
manual(19) = 10; %x
manual(20) = 10; %y
manual(21) = 10; %z
manual(22) = 5; %V
manual(23) = 10; %theta
manual(24) = 10; %phi
manual(25) = 0; %dx
manual(26) = 0; %dy
manual(27) = 0; %dz
manual(28) = 0; %dV

```

```

manual(29) = 0; %dtheta
manual(30) = 0; %dphi

manual(31) = 500; %p_num

manual(32) = .1; %x variance
manual(33) = .1; %y variance
manual(34) = .1; %z variance
manual(35) = .5; %V variance
manual(36) = .5; %theta variance
manual(37) = .5; %phi variance
manual(38) = .1; %dxp variance
manual(39) = .1; %dyp variance
manual(40) = .1; %dzp variance
manual(41) = .1; %dVp variance
manual(42) = .1; %dthetap variance
manual(43) = .1; %dphip variance

manual(44) = .1; %Measurement noise covariance

manual(45) = 1; %Filter1 u weight
manual(46) = 1; %Filter1 v weight
manual(47) = 1; %Filter1 w weight

%EPF-B
manual(48) = 10; %x
manual(49) = 10; %y
manual(50) = 10; %z
manual(51) = 5; %V
manual(52) = 10; %theta
manual(53) = 10; %phi
manual(54) = 0; %dx

```

```

manual(55) = 0; %dy
manual(56) = 0; %dz
manual(57) = 0; %dV
manual(58) = 0; %dtheta
manual(59) = 0; %dphi

manual(60) = 1000; %p_num

manual(61) = .1; %x variance
manual(62) = .1; %y variance
manual(63) = .1; %z variance
manual(64) = .5; %V variance
manual(65) = .5; %theta variance
manual(66) = .5; %phi variance
manual(67) = .1; %dyp variance
manual(68) = .1; %dyp variance
manual(69) = .1; %dzp variance
manual(70) = .1; %dVp variance
manual(71) = .1; %dthetap variance
manual(72) = .1; %dphip variance

manual(73) = 30; %Measurement noise covariance

manual(74) = 1; %6 %1 %100 (to see if this affects the weight value)
manual(75) = 1; %6 %1
manual(76) = 1; %3 %10
manual(77) = 1; %4 %10
manual(78) = 1; %4 %10

switch target_scenario
    case 1

```

```

x=1; y=0; z=0; V=4; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 2
x=0; y=1; z=0; V=4; theta=0; phi=0; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 3
x=0; y=0; z=10; V=4; theta=90; phi=90; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 4
x=1; y=0; z=0; V=4; theta=90; phi=180; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 5
x=0; y=1; z=0; V=4; theta=180; phi=0; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 6
x=0; y=0; z=10; V=4; theta=90; phi=270; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=0; xdot=0; ydot=0; zdot=0;
case 7
x=0; y=0; z=20; V=5; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
dtheta=1; dphi=0; xdot=0; ydot=0; zdot=0;
case 8
x=0; y=0; z=10; V=5; theta=90; phi=0; dx=0; dy=0; dz=0; dV=0;...
dtheta=0; dphi=1; xdot=0; ydot=0; zdot=0;
case 9
x=0; y=0; z=10; V=5; theta=0; phi=90; dx=0; dy=0; dz=0; dV=0;...
dtheta=1; dphi=0; xdot=0; ydot=0; zdot=0;
case 10
x=10; y=10; z=10; V=5; theta=10; phi=10; dx=0; dy=0; dz=0;...
dV=.2; dtheta=.5; dphi=.5; xdot=0; ydot=0; zdot=0;
case 0
x>manual(1); y>manual(2); z>manual(3); V>manual(4);...
theta>manual(5);

```

```

        phi>manual(6); dx>manual(7); dy>manual(8); dz>manual(9);...
        dV>manual(10);
        dtheta>manual(11); dphi>manual(12); xdot>manual(13);...
        ydot>manual(14); zdot>manual(15);
    otherwise
        error('INCORRECT TARGET SCENARIO SELECTION')
end

switch P_variance_scenario
    case 1
        TProcess_V=.1; TProcess_theta=deg2rad(5); TProcess_phi=deg2rad(5);
    case 2
        TProcess_V=.01; TProcess_theta=.01; TProcess_phi=.01;
    case 0
        TProcess_V=0; TProcess_theta=0; TProcess_phi=0;
    otherwise
        error('INCORRECT PROCESS VARIANCE SCENARIO SELECTION')
end

switch m_variance_scenario
    case 1
        Mu_var=0; Mv_var=0; Mw_var=0;
    case 2
        Mu_var=5; Mv_var=5; Mw_var=5;
    case 3
        Mu_var=10; Mv_var=10; Mw_var=10;
    case 0
        Mu_var>manual(16); Mv_var>manual(17); Mw_var>manual(18);
    otherwise
        error('INCORRECT VARIANCE SCENARIO SELECTION')
end

```

```

switch filter1_scenario
    case 1
        xp1=x; yp1=y; zp1=z; Vp1=V; thetap1=theta; phip1=phi;
        dxp1=dx; dyp1=dy; dzp1=dz; dVp1=dV; dthetap1=dtheta; dhip1=dphi;
    case 2
        xp1=0; yp1=0; zp1=0; Vp1=0; thetap1=0; phip1=0;
        dxp1=0; dyp1=0; dzp1=0; dVp1=0; dthetap1=0; dhip1=0;
    case 3
        xp1=0; yp1=0; zp1=10; Vp1=0; thetap1=0; phip1=0;
        dxp1=0; dyp1=0; dzp1=0; dVp1=0; dthetap1=0; dhip1=0;
    case 0
        xp1>manual(19); yp1>manual(20); zp1>manual(21);
        Vp1>manual(22); thetap1>manual(23); phip1>manual(24);
        dxp1>manual(25); dyp1>manual(26); dzp1>manual(27);
        dVp1>manual(28); dthetap1>manual(29); dhip1>manual(30);
    otherwise
        error('INCORRECT FILTER1 INTIAL CONDITIONS')
end

switch filter2_scenario
    case 1
        xp2=x; yp2=y; zp2=z; Vp2=V; thetap2=theta; phip2=phi;
        dxp2=dx; dyp2=dy; dzp2=dz; dVp2=dV; dthetap2=dtheta; dhip2=dphi;
    case 2
        xp2=0; yp2=0; zp2=0; Vp2=0; thetap2=0; phip2=0;
        dxp2=0; dyp2=0; dzp2=0; dVp2=0; dthetap2=0; dhip2=0;
    case 3
        xp2=0; yp2=0; zp2=10; Vp2=0; thetap2=0; phip2=0;
        dxp2=0; dyp2=0; dzp2=0; dVp2=0; dthetap2=0; dhip2=0;
    case 0
        xp2>manual(48); yp2>manual(49); zp2>manual(50);
        Vp2>manual(51); thetap2>manual(52); phip2>manual(53);

```

```

dxp2>manual(54); dyp2>manual(55); dzp2>manual(56);
dVp2>manual(57); dthetap2>manual(58); dhip2>manual(59);
otherwise
    error('INCORRECT FILTER1 INTIAL CONDITIONS')
end

switch filter1_variance
case 1
    xp1_v=1; yp1_v=1; zp1_v=1; Vp1_v=.5; thetap1_v=.1; phip1_v=.1;
    dxp1_v=.5; dyp1_v=.5; dzp1_v=.5; dVp1_v=.1; dthetap1_v=.01;...
    dhip1_v=.01;
case 0
    xp1_v>manual(32); yp1_v>manual(33); zp1_v>manual(34);
    Vp1_v>manual(35); thetap1_v>manual(36); phip1_v>manual(37);
    dxp1_v>manual(38); dyp1_v>manual(39); dzp1_v>manual(40);
    dVp1_v>manual(41); dthetap1_v>manual(42); dhip1_v>manual(43);
otherwise
    error('INCORRECT FILTER1 STATE VARIANCES')
end

switch filter2_variance
case 1
    xp2_v=1; yp2_v=1; zp2_v=1; Vp2_v=.5; thetap2_v=.1; phip2_v=.1;
    dxp2_v=.5; dyp2_v=.5; dzp2_v=.5; dVp2_v=.1; dthetap2_v=.01;...
    dhip2_v=.01;
case 2
    xp2_v=.1; yp2_v=.1; zp2_v=.1; Vp2_v=20; thetap2_v=4; phip2_v=4;
    dxp2_v=.01; dyp2_v=.01; dzp2_v=.01; dVp2_v=.01; dthetap2_v=.01;...
    dhip2_v=.01;
case 0
    xp2_v>manual(61); yp2_v>manual(62); zp2_v>manual(63);
    Vp2_v>manual(64); thetap2_v>manual(65); phip2_v>manual(66);

```

```

        dxp2_v>manual(67); dyp2_v>manual(68); dzp2_v>manual(69);
        dVp2_v>manual(70); dthetap2_v>manual(71); dphip2_v>manual(72);
    otherwise
        error('INCORRECT FILTER1 STATE VARIANCES')
end

switch filter1_measurement_variance
    case 1
        m1_v=Mu_var;
    case 2
        m1_v=.01;
    case 3
        m1_v=.1;
    case 0
        m1_v>manual(44);
    otherwise
        error('INCORRECT FILTER1 MEASUREMENT VARIANCES')
end

switch filter2_measurement_variance
    case 1
        m2_v=Mu_var;
    case 2
        m2_v=.01;
    case 3
        m2_v=30;
    case 0
        m2_v>manual(73);
    otherwise
        error('INCORRECT FILTER2 MEASUREMENT VARIANCES')
end

```

```

switch filter1.weight
    case 1
        W_u=1; W_v=1; W_w=1;
    case 0
        W_u>manual(45); W_v>manual(46); W_w>manual(47);
    otherwise
        error('INCORRECT FILTER1 WEIGHTS')
end

switch filter2.weight
    case 1
        W_up=1; W_vp=1; W_s=1; W_up_dot=1; W_vp_dot=1;
    case 0
        W_up>manual(74); W_vp>manual(75); W_s>manual(76);...
        W_up_dot>manual(77); W_vp_dot>manual(78);
    otherwise
        error('INCORRECT FILTER2 WEIGHTS')
end

target(1)    =    x; %x
target(2)    =    y; %y
target(3)    =    z; %z
target(4)    =    V;  %V
target(5)    =    theta; %theta
target(6)    =    phi; %phi
target(7)    =    dx;  %dx
target(8)    =    dy;  %dy
target(9)    =    dz;  %dz
target(10)   =    dV;  %dV
target(11)   =    dtheta; %dtheta
target(12)   =    dphi; %dphi
target(13)   =    xdot; %x_dot

```

```

target(14) = ydot; %y_dot
target(15) = zdot; %z_dot
target(16) = TProcess_V;
target(17) = TProcess_theta;
target(18) = TProcess_phi;

obs(1) = Mu_var;
obs(2) = Mv_var;
obs(3) = Mw_var;

%EPF-A
filter1(1) = xp1; %x
filter1(2) = yp1; %y
filter1(3) = zp1; %z
filter1(4) = Vp1; %V
filter1(5) = thetap1; %
filter1(6) = phip1;
filter1(7) = dxp1;
filter1(8) = dyp1;
filter1(9) = dzp1;
filter1(10) = dVp1;
filter1(11) = dthetap1;
filter1(12) = dhip1;

filter1(13) = p_num;

filter1(14) = xp1_v; %x variance
filter1(15) = yp1_v; %y variance
filter1(16) = zp1_v; %z variance
filter1(17) = Vp1_v; %V variance
filter1(18) = thetap1_v; %theta variance
filter1(19) = phip1_v; %phi variance

```

```

filter1(20) = dxp1_v; %dyp variance
filter1(21) = dyp1_v; %dyp variance
filter1(22) = dzp1_v; %dzp variance
filter1(23) = dVp1_v; %dVp variance
filter1(24) = dthetap1_v; %dthetap variance
filter1(25) = dhip1_v; %dhip variance

filter1(26) = m1_v; %Measurement noise covariance

filter1(27) = W_u;
filter1(28) = W_v;
filter1(29) = W_w;

%EPF-B
filter2(1) = xp2; %x
filter2(2) = yp2; %y
filter2(3) = zp2; %z
filter2(4) = Vp2; %V
filter2(5) = thetap2; %
filter2(6) = phip2;
filter2(7) = dxp2;
filter2(8) = dyp2;
filter2(9) = dzp2;
filter2(10) = dVp2;
filter2(11) = dthetap2;
filter2(12) = dhip2;

filter2(13) = p2_num;

filter2(14) = xp2_v; %x variance
filter2(15) = yp2_v; %y variance
filter2(16) = zp2_v; %z variance

```

```

filter2(17) = Vp2_v; %V variance
filter2(18) = thetap2_v; %theta variance
filter2(19) = phip2_v; %phi variance
filter2(20) = dxp2_v; %dyp variance
filter2(21) = dyp2_v; %dyp variance
filter2(22) = dzp2_v; %dzp variance
filter2(23) = dVp2_v; %dVp variance
filter2(24) = dthetap2_v; %dthetap variance
filter2(25) = dhip2_v; %dhip variance

filter2(26) = m2_v; %Measurement noise covariance

filter2(27) = W_up;
filter2(28) = W_vp;
filter2(29) = W_s;
filter2(30) = W_up_dot;
filter2(31) = W_vp_dot;

err=0; %Number of weight discrepancy crashes
%Preallocation
%sim_T_G_cent=zeros(params(1),12,sim_run);
sim_O_G_cent=zeros(params(1),6,sim_run);
sim_P_G_S=zeros(params(1),15,sim_run);
sim_C1_G_cent=zeros(params(1),9,sim_run);
sim_P2_G_S=zeros(params(1),15,sim_run);
sim_P2_G_S_part_u=zeros(p2_num,15,params(1),sim_run);
sim_P2_P_O_part=zeros(p2_num,5,params(1),sim_run);
sim_P2_P_W=zeros(params(1),p2_num,sim_run);
sim_C2_G_cent=zeros(params(1),9,sim_run);

c=1;
while c<=sim_run

```

```

[T_G.cent, T_P.cent, O_G.cent, P_G.S, P2_G.S, P2_G.S.part_u,...
    P2_P.O.part, P2_P.W, C1_G.cent, C2_G.cent, flag] =...
    EPF_B.Function.Final(params, target, obs, filter1, filter2,...
        correction);

%Convert to degrees
if flag == 0
    T_G.cent(:, [5,6,11,12])=rad2deg(T_G.cent(:, [5,6,11,12]));
    P_G.S(:, [5,6,11,12])=rad2deg(P_G.S(:, [5,6,11,12]));
    C1_G.cent(:, [8,9])=rad2deg(C1_G.cent(:, [8,9]));

    P2_G.S(:, [5,6,11,12])=rad2deg(P2_G.S(:, [5,6,11,12]));
    C2_G.cent(:, [8,9])=rad2deg(C2_G.cent(:, [8,9]));

    sim_T_G.cent(:, :, c)=T_G.cent;
    sim_O_G.cent(:, :, c)=O_G.cent;
    sim_P_G.S(:, :, c)=P_G.S;
    sim_C1_G.cent(:, :, c)=C1_G.cent;
    sim_P2_G.S(:, :, c)=P2_G.S;
    sim_P2_G.S.part_u(:, :, :, c)=P2_G.S.part_u;
    sim_P2_P.O.part(:, :, :, c)=P2_P.O.part;
    sim_P2_P.W(:, :, c)=P2_P.W;
    sim_C2_G.cent(:, :, c)=C2_G.cent;
    c=c+1;
end

if flag == 1;
    message='WEIGHT DISCREPANCY';
    err=err+1;
end

c
err

```

```

end
err

switch sim_plot
case 1
    fig=1;
    hfig=figure(fig);
    set(hfig,'Position',[0, 0, 800, 800]); %[x y width height]
    axis equal
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    for i2=4:params(1)
        hold on;
        scatter3(T_G_cent(i2,1),T_G_cent(i2,2),T_G_cent(i2,3),'b');
        scatter3(O_G_cent(i2,4),O_G_cent(i2,5),O_G_cent(i2,6),'g');
        scatter3(C1_G_cent(i2,1),C1_G_cent(i2,2),C1_G_cent(i2,3),'m');
        scatter3(P_G_S(i2,1),P_G_S(i2,2),P_G_S(i2,3),'r');
        scatter3(P2_G_S(i2,1),P2_G_S(i2,2),P2_G_S(i2,3),'r+');
        scatter3(C2_G_cent(i2,1),C2_G_cent(i2,2),C2_G_cent(i2,3),'m+');
        pause(.1)
    end
    legend('Target','Measurement','Comparison','Filter',...
        'Location','East')
    fig=fig+1;

    ang_red=@(a) atan2(sin(a),cos(a));

    for i=4:params(1)
        figure(fig);clf;
        set(gcf,'position',[ 680,72, 1152, 906]);
        % Plot weighted particle distribution for diagnostic purposes

```

```

a=subplot(3,2,1);
stem3(P2_P_O_part(:,1,i),P2_P_O_part(:,2,i),P2_P_W(i,:), 'ro');
hold on; stem3(T_P_cent(i,1),T_P_cent(i,2),...
    max(P2_P_W(i,:), 'k*'));
title(a, 'U vs V Position Particle Weight');
%    aspect([1e1,1e1,1e-2]);drawnow;
a=subplot(3,2,3);
stem(P2_P_O_part(:,3,i),P2_P_W(i,:), 'ro');
hold on; stem(T_P_cent(i,3),max(P2_P_W(i,:), 'k*'));
title(a, 'Z Position Particle Weight');
%    aspect([1e-4,1e-2,1]);drawnow;
a=subplot(3,2,5);
stem3(P2_P_O_part(:,4,i),P2_P_O_part(:,5,i),P2_P_W(i,:), 'ro');
hold on; stem3(T_P_cent(i,4),T_P_cent(i,5),...
    max(P2_P_W(i,:), 'k*'));
title(a, 'U vs V Velocity Particle Weight');
%    aspect([1e2,1e2,1e-2]);drawnow;

a=subplot(3,2,2);
stem3(P2_G_S_part_u(:,1,i),P2_G_S_part_u(:,2,i),...
    P2_P_W(i,:), 'ro');
hold on; stem3(T_G_cent(i,1),T_G_cent(i,2),...
    max(P2_P_W(i,:), 'k*'));
title(a, 'X vs Y Position Particle Weight');
%    aspect([1e1,1e1,1e-2]);drawnow;
a=subplot(3,2,4);
stem3(P2_G_S_part_u(:,3,i),P2_G_S_part_u(:,4,i),...
    P2_P_W(i,:), 'ro');
hold on; stem3(T_G_cent(i,3),T_G_cent(i,4),...
    max(P2_P_W(i,:), 'k*'));
title(a, 'Z Position Particle Weight');
%    aspect([1e1,1e1,1e-2]);drawnow;

```

```

a=subplot(3,2,6);
stem3(ang_red(P2_G_S_part_u(:,5,i)),...
      ang_red(P2_G_S_part_u(:,6,i)),P2_P_W(i,:), 'ro');
hold on; stem3(ang_red(T_G_cent(i,3)),...
      ang_red(T_G_cent(i,4)),max(P2_P_W(i,:),'k*');
title(a, 'X vs Y Velocity Particle Weight');
pause();
end

```

```

for i=4:params(1)
    %Weight Scatter Plots
    figure(fig);clf;
    set(gcf, 'position', [0, 0, 800, 1200]);
    a=subplot(3,2,1);
    stem3(P2_P_O_part(:,1,i),P2_P_O_part(:,2,i),...
          P2_P_W(i,:), 'ro');
    hold on; stem3(T_P_cent(i,1),T_P_cent(i,2),...
          max(P2_P_W(i,:),'k*');
    title(a, 'U vs V Position Particle Weight');
    xlabel(a, 'u');ylabel(a, 'v');

    a=subplot(3,2,3);
    stem(P2_P_O_part(:,3,i),P2_P_W(i,:), 'ro');
    hold on; stem(T_P_cent(i,3),max(P2_P_W(i,:),'k*');
    title(a, 'W Position Particle Weight');
    xlabel(a, 'looming');

    a=subplot(3,2,5);
    stem3(P2_P_O_part(:,4,i),P2_P_O_part(:,5,i),...
          P2_P_W(i,:), 'ro');
    hold on; stem3(T_P_cent(i,4),T_P_cent(i,5),...

```

```

        max(P2_P_W(i,:), 'k*');
title(a, 'U vs V Velocity Particle Weight');
        xlabel(a, 'u_d_o_t'); ylabel(a, 'v_d_o_t');

a=subplot(3,2,2);
temp_r=T_G_cent(i,1:3) ./ norm(T_G_cent(i,1:3));
stem(P2_G_S_part_u(:,1:3,i)*temp_r, P2_P_W(i,:), 'ro');
hold on; stem(T_G_cent(i,1:3)*temp_r, max(P2_P_W(i,:), 'k*'));
title(a, 'Position Particle Weight');
        xlabel(a, 'r');

a=subplot(3,2,4);
stem(P2_G_S_part_u(:,4,i), P2_P_W(i,:), 'ro');
hold on; stem(T_G_cent(i,4), max(P2_P_W(i,:), 'k*'));
title(a, 'Velocity Particle Weight');
        xlabel(a, 'V');

a=subplot(3,2,6);
stem3(ang_red(P2_G_S_part_u(:,5,i)), ...
        ang_red(P2_G_S_part_u(:,6,i)), P2_P_W(i,:), 'ro');
hold on; stem3(ang_red(T_G_cent(i,5)), ...
        ang_red(T_G_cent(i,6)), max(P2_P_W(i,:), 'k*'));
title(a, 'X vs Y Velocity Particle Weight');
        xlabel(a, '\theta'); ylabel(a, '\phi');
        pause(.1);
end

%Comparison of States
time=zeros(params(1),1);
for i2=3:params(1)
        time(i2,1)=i2;
end

```

```

%Positions
fig=fig+1;
hfig=figure(fig);
set(hfig,'Position',[0, 0, 800, 1200]); %[x y width height]
a=subplot(3,1,1);
plot(time,T_G_cent(:,1),'b',...
      time,P_G_S(:,1),'r',...
      time,P2_G_S(:,1),'r:',...
      time,C1_G_cent(:,1),'g',...
      time,C2_G_cent(:,1),'g:');
legend('Target','Filter 1','Filter 2','Comparison 1',...
       'Comparison 2','Location','SouthEast')
title(a,'X Position');

b=subplot(3,1,2);
plot(time,T_G_cent(:,2),'b',...
      time,P_G_S(:,2),'r',...
      time,P2_G_S(:,2),'r:',...
      time,C1_G_cent(:,2),'g',...
      time,C2_G_cent(:,2),'g:');
legend('Target','Filter 1','Filter 2','Comparison 1',...
       'Comparison 2','Location','SouthEast')
title(b,'Y Position');

c=subplot(3,1,3);
plot(time,T_G_cent(:,3),'b',...
      time,P_G_S(:,3),'r',...
      time,P2_G_S(:,3),'r:',...
      time,C1_G_cent(:,3),'g',...
      time,C2_G_cent(:,3),'g:');
legend('Target','Filter 1','Filter 2','Comparison 1',...

```

```

        'Comparison 2','Location','SouthEast')
title(c, 'Z Position');
fig=fig+1;

%Velocity Magintude and headings
hfig=figure(fig);
set(hfig, 'Position', [0, 0, 800, 1200]); %[x y width height]
a=subplot(3,1,1);
plot(time, T-G-cent(:,4), 'b', ...
      time, P-G-S(:,4), 'r', ...
      time, P2-G-S(:,4), 'r:', ...
      time, C1-G-cent(:,7), 'g', ...
      time, C2-G-cent(:,7), 'g:');
legend('Target', 'Filter 1', 'Filter 2', 'Comparison 1', ...
       'Comparison 2', 'Location', 'SouthEast')
title(a, 'Velocity Magnitude');

b=subplot(3,1,2);
plot(time, (T-G-cent(:,5)), 'b', ...
      time, (P-G-S(:,5)), 'r', ...
      time, (P2-G-S(:,5)), 'r:', ...
      time, (C1-G-cent(:,8)), 'g', ...
      time, (C2-G-cent(:,8)), 'g:');
legend('Target', 'Filter 1', 'Filter 2', 'Comparison 1', ...
       'Comparison 2', 'Location', 'SouthEast')
title(b, 'Theta Heading');

c=subplot(3,1,3);
plot(time, (T-G-cent(:,6)), 'b', ...
      time, (P-G-S(:,6)), 'r', ...
      time, (P2-G-S(:,6)), 'r:', ...
      time, (C1-G-cent(:,9)), 'g', ...

```

```

        time, (C2_G_cent(:,9)), 'g:');
legend('Target', 'Filter 1', 'Filter 2', 'Comparison 1', ...
       'Comparison 2', 'Location', 'SouthEast')
title(c, 'Phi Heading');
fig=fig+1;

otherwise

    message='WILL NOT PLOT, MULTIPLE SIMULATION RUNS';
    err

%% Calculate performace abilities

    %%NOTE: Although distance and V errors are absolute, heading errors
    %can only be up to 180 degrees off for both phi and theta (ie. you
    %are pointinig in the complete opposite direction. Theta is
    %already between 0 and 180, so aboslute errors may be taken. Phi
    %though must be adjusted so that all errors are between 0 and 180
    %(ex. an 'error' of 350 is really only an error of 10

    %Preallocate
    error_m_f=zeros(params(1),4);
    error_m_c=zeros(params(1),4);
    vec_sim_P_G_S=zeros(params(1),4);
    vec_sim_C1_G_cent=zeros(params(1),4);

    %'Best' Values for filter and comparison
    range_index=solution_range*sim_run;
    for i=2:params(1)
        for i2=1:sim_run

            %Angle Adjustment
            phi=sim_P2_G_S(i,6,i2);
            if (phi > 360)

```

```

        mult=floor(phi/(360));
        phi=phi-mult*360;
end

if (phi < -360)
    mult=floor(phi/(-360));
    phi=phi+mult*360;
end

if (phi < 0)
    phi=360+phi;
end

sim_P2_G_S(i,6,i2)=abs(phi);

theta=sim_P2_G_S(i,5,i2);
theta=wrapTo2Pi(theta);
if(theta>pi())
    theta_a=theta-pi();
    theta=pi()-theta_a;
end

sim_P2_G_S(i,5,i2)=theta;

%Create variable vectors
t_dist=sqrt(sim_T_G_cent(i,1,i2)^2+...
    sim_T_G_cent(i,2,i2)^2+sim_T_G_cent(i,2,i2)^2);
f_dist=sqrt(sim_P_G_S(i,1,i2)^2+sim_P_G_S(i,2,i2)^2+...
    sim_P_G_S(i,2,i2)^2);
c_dist=sqrt(sim_C1_G_cent(i,1,i2)^2+...
    sim_C1_G_cent(i,2,i2)^2+sim_C1_G_cent(i,2,i2)^2);

f2_dist=sqrt(sim_P2_G_S(i,1,i2)^2+...
    sim_P2_G_S(i,2,i2)^2+sim_P2_G_S(i,2,i2)^2);

```

```

c2_dist=sqrt(sim_C2_G_cent(i,1,i2)^2+...
    sim_C2_G_cent(i,2,i2)^2+sim_C2_G_cent(i,2,i2)^2);

vec_sim_P_G_S(i2,1)=abs(f_dist-t_dist);
vec_sim_P_G_S(i2,[2,3,4])=abs(sim_P_G_S(i,[4,5,6],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));
vec_sim_C1_G_cent(i2,1)=abs(c_dist-t_dist);
vec_sim_C1_G_cent(i2,[2,3,4])=...
    abs(sim_C1_G_cent(i,[7,8,9])-...
    sim_T_G_cent(i,[4,5,6],i2));

vec_sim_P2_G_S(i2,1)=abs(f2_dist-t_dist);
vec_sim_P2_G_S(i2,[2,3,4])=abs(sim_P2_G_S(i,[4,5,6],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));
vec_sim_C2_G_cent(i2,1)=abs(c2_dist-t_dist);
vec_sim_C2_G_cent(i2,[2,3,4])=...
    abs(sim_C2_G_cent(i,[7,8,9],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));

%Phi adjustment
if vec_sim_P_G_S(i2,4)>180
    vec_sim_P_G_S(i2,4)=360-vec_sim_P_G_S(i2,4);
end

if vec_sim_C1_G_cent(i2,4)>180
    vec_sim_C1_G_cent(i2,4)=360-vec_sim_C1_G_cent(i2,4);
end

if vec_sim_P2_G_S(i2,4)>180
    vec_sim_P2_G_S(i2,4)=360-vec_sim_P2_G_S(i2,4);
end

```

```

        if vec_sim_C2_G_cent(i2,4)>180
            vec_sim_C2_G_cent(i2,4)=360-vec_sim_C2_G_cent(i2,4);
        end
    end

    svec_sim_P_G_S=sort(vec_sim_P_G_S,1);
    svec_sim_C1_G_cent=sort(vec_sim_C1_G_cent,1);
    error_b_f(i,:)=svec_sim_P_G_S(range_index,:);
    error_b_c(i,:)=svec_sim_C1_G_cent(range_index,:);

    svec_sim_P2_G_S=sort(vec_sim_P2_G_S,1);
    svec_sim_C2_G_cent=sort(vec_sim_C2_G_cent,1);
    error_b_f2(i,:)=svec_sim_P2_G_S(range_index,:);
    error_b_c2(i,:)=svec_sim_C2_G_cent(range_index,:);
end

%Mean or average error values for filter and comparison
for i=2:params(1)
    for i2=1:sim_run
        %Create variable vectors
        t_dist=sqrt(sim_T_G_cent(i,1,i2)^2+...
            sim_T_G_cent(i,2,i2)^2+sim_T_G_cent(i,3,i2)^2);
        f_dist=sqrt(sim_P_G_S(i,1,i2)^2+sim_P_G_S(i,2,i2)^2+...
            sim_P_G_S(i,3,i2)^2);
        c_dist=sqrt(sim_C1_G_cent(i,1,i2)^2+...
            sim_C1_G_cent(i,2,i2)^2+sim_C1_G_cent(i,3,i2)^2);

        f2_dist=sqrt(sim_P2_G_S(i,1,i2)^2+sim_P2_G_S(i,2,i2)^2+...
            sim_P2_G_S(i,3,i2)^2);
        c2_dist=sqrt(sim_C2_G_cent(i,1,i2)^2+...
            sim_C2_G_cent(i,2,i2)^2+sim_C2_G_cent(i,3,i2)^2);
    end
end

```

```

vec_sim_P_G_S(i2,1)=abs(f_dist-t_dist);
vec_sim_P_G_S(i2,[2,3,4])=abs(sim_P_G_S(i,[4,5,6],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));
vec_sim_C1_G_cent(i2,1)=abs(c_dist-t_dist);
vec_sim_C1_G_cent(i2,[2,3,4])=...
    abs(sim_C1_G_cent(i,[7,8,9],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));

vec_sim_P2_G_S(i2,1)=abs(f2_dist-t_dist);
vec_sim_P2_G_S(i2,[2,3,4])=abs(sim_P2_G_S(i,[4,5,6],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));
vec_sim_C2_G_cent(i2,1)=abs(c2_dist-t_dist);
vec_sim_C2_G_cent(i2,[2,3,4])=...
    abs(sim_C2_G_cent(i,[7,8,9],i2)-...
    sim_T_G_cent(i,[4,5,6],i2));

if vec_sim_P_G_S(i2,4)>180
    vec_sim_P_G_S(i2,4)=360-vec_sim_P_G_S(i2,4);
end

if vec_sim_C1_G_cent(i2,4)>180
    vec_sim_C1_G_cent(i2,4)=360-vec_sim_C1_G_cent(i2,4);
end

if vec_sim_P2_G_S(i2,4)>180
    vec_sim_P2_G_S(i2,4)=360-vec_sim_P2_G_S(i2,4);
end

if vec_sim_C2_G_cent(i2,4)>180
    vec_sim_C2_G_cent(i2,4)=360-vec_sim_C2_G_cent(i2,4);
end
end
end

```

```

    error_m_f(i,1)=mean(vec_sim_P_G_S(:,1));
    error_m_f(i,2)=mean(vec_sim_P_G_S(:,2));
    error_m_f(i,3)=mean(vec_sim_P_G_S(:,3));
    error_m_f(i,4)=mean(vec_sim_P_G_S(:,4));

    error_m_c(i,1)=mean(vec_sim_C1_G_cent(:,1));
    error_m_c(i,2)=mean(vec_sim_C1_G_cent(:,2));
    error_m_c(i,3)=mean(vec_sim_C1_G_cent(:,3));
    error_m_c(i,4)=mean(vec_sim_C1_G_cent(:,4));

    error_m_f2(i,1)=mean(vec_sim_P2_G_S(:,1));
    error_m_f2(i,2)=mean(vec_sim_P2_G_S(:,2));
    error_m_f2(i,3)=mean(vec_sim_P2_G_S(:,3));
    error_m_f2(i,4)=mean(vec_sim_P2_G_S(:,4));

    error_m_c2(i,1)=mean(vec_sim_C2_G_cent(:,1));
    error_m_c2(i,2)=mean(vec_sim_C2_G_cent(:,2));
    error_m_c2(i,3)=mean(vec_sim_C2_G_cent(:,3));
    error_m_c2(i,4)=mean(vec_sim_C2_G_cent(:,4));
end

time=zeros(params(1),1);
for i2=2:params(1)
    time(i2,1)=i2;
end

fig=1;
%TER
hfig=figure(fig);
set(hfig,'Position',[0, 0, 800, 1200]); %[x y width height]
a=subplot(4,1,1);
plot(time,error_b_f(:,1),'m',...

```

```

        time,error_b_f2(:,1),'r:',...
        time,error_b_c(:,1),'g',...
        time,error_b_c2(:,1),'g:');
legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(a,'Distance Error');
xlabel(a,'Time Step');
ylabel(a,'Error');

b=subplot(4,1,2);
plot(time,error_b_f(:,2),'m',...
        time,error_b_f2(:,2),'r:',...
        time,error_b_c(:,2),'g',...
        time,error_b_c2(:,2),'g:',...
        time,T_G_cent(:,4),'b');

legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(b,'Velocity Error');
xlabel(b,'Time Step');
ylabel(b,'Error');

c=subplot(4,1,3);
plot(time,error_b_f(:,3),'m',...
        time,error_b_f2(:,3),'r:',...
        time,error_b_c(:,3),'g',...
        time,error_b_c2(:,3),'g:',...
        time,T_G_cent(:,5),'b');
legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(c,'Theta Heading Error');
xlabel(c,'Time Step');

```

```

ylabel(c, 'Error (Deg)');

d=subplot(4,1,4);
plot(time,error_b_f(:,4), 'm', ...
      time,error_b_f2(:,4), 'r:', ...
      time,error_b_c(:,4), 'g', ...
      time,error_b_c2(:,4), 'g:', ...
      time,T_G_cent(:,6), 'b');
legend('Filter 1', 'Filter 2', 'Comparison 1', 'Comparison 2', ...
       'Target', 'Location', 'North', 'Orientation', 'horizontal')
title(d, 'Phi Heading Error');
xlabel(d, 'Time Step');
ylabel(d, 'Error (Deg)');
fig=fig+1;

%MAE
hfig=figure(fig);
set(hfig, 'Position', [0, 0, 800, 1200]); %[x y width height]
a=subplot(4,1,1);
plot(time,error_m_f(:,1), 'm', ...
      time,error_m_f2(:,1), 'r:', ...
      time,error_m_c(:,1), 'g', ...
      time,error_m_c2(:,1), 'g:');
legend('Filter 1', 'Filter 2', 'Comparison 1', 'Comparison 2', ...
       'Target', 'Location', 'North', 'Orientation', 'horizontal')
title(a, 'Distance Error');
xlabel(a, 'Time Step');
ylabel(a, 'Error');

b=subplot(4,1,2);
plot(time,error_m_f(:,2), 'm', ...
      time,error_m_f2(:,2), 'r:', ...

```

```

        time,error_m_c(:,2),'g',...
        time,error_m_c2(:,2),'g:',...
        time,T_G_cent(:,4),'b');
legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(b, 'Velocity Error');
xlabel(b, 'Time Step');
ylabel(b, 'Error');

c=subplot(4,1,3);
plot(time,error_m_f(:,3),'m',...
        time,error_m_f2(:,3),'r:',...
        time,error_m_c(:,3),'g',...
        time,error_m_c2(:,3),'g:',...
        time,T_G_cent(:,5),'b');
legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(c, 'Theta Heading Error');
xlabel(c, 'Time Step');
ylabel(c, 'Error (Deg)');

d=subplot(4,1,4);
plot(time,error_m_f(:,4),'m',...
        time,error_m_f2(:,4),'r:',...
        time,error_m_c(:,4),'g',...
        time,error_m_c2(:,4),'g:',...
        time,T_G_cent(:,6),'b');
legend('Filter 1','Filter 2','Comparison 1','Comparison 2',...
        'Target','Location','North','Orientation','horizontal')
title(d, 'Phi Heading Error');
xlabel(d, 'Time Step');
ylabel(d, 'Error (Deg)');

```

```

fig=fig+1;

%Final 50 step metrics
best_error_c=mean(error_t_c(params(1)-50:params(1),:))
best_error_c=mean(error_t_c2(params(1)-50:params(1),:))
best_error_f=mean(error_t_f(params(1)-50:params(1),:))
best_error_f=mean(error_t_f2(params(1)-50:params(1),:))

%Final 50 step metrics
mean_error_c=mean(error_m_c(params(1)-50:params(1),:))
mean_error_c=mean(error_m_c2(params(1)-50:params(1),:))
mean_error_f=mean(error_m_f(params(1)-50:params(1),:))
mean_error_f=mean(error_m_f2(params(1)-50:params(1),:))
end

```

A.5.2 EPF B Function Final.

```

function [T_G_cent, T_P_cent, O_G_cent, P_G_S, P2_G_S, P2_G_S_part_u,...
    P2_P_O_part, P2_P_W, L1_G_cent, L2_G_cent, flag] =...
    EPF_B.Function.Final(params, target, obs, filter1, filter2, correction)
%% This function is Particle Filter A, allowing for multiple scenarios or
%% simulation runs. Inputs are contained within a separate .mfile that
%% executes this file.

%% Initial conditions and setup
rng('shuffle') %Shuffle random numbers
%System Parameters
T    =    params(1); %Number of iterations
dt   =    params(2); %Time step
f    =    params(3); %Focal Length
flag=0;

```

```

correct=correction;

%Rotation matrix between global and camera frames
Rot_y=@(a) [cos(a), 0, -sin(a); 0,1,0; sin(a), 0, cos(a)];
Rot_x=@(a) [1,0,0;0,cos(a), -sin(a); 0,sin(a), cos(a)];
Rot_z=@(a) [cos(a),-sin(a),0;sin(a), cos(a),0;0,0,1];
PlotDCM=@(A,O) plot3(cumsum([O(1),A(1,1)]),cumsum([O(2),A(2,1)]),...
    cumsum([O(3),A(3,1)]),'r-',...
    cumsum([O(1),A(1,2)]),cumsum([O(2),A(2,2)]),cumsum([O(3),A(3,2)]),'g-',...
    cumsum([O(1),A(1,3)]),cumsum([O(2),A(2,3)]),cumsum([O(3),A(3,3)]),'b-',...
    'linewidth',5);
ang_red=@(a) atan2(sin(a),cos(a));

%% Target Parameters
%Truth conditions, the initial conditions of the centroid
x = target(1);
y = target(2);
z = target(3);
V = target(4);
theta = deg2rad(target(5));
phi = deg2rad(target(6));
dx = target(7);
dy = target(8);
dz = target(9);
dV = target(10);
dtheta = deg2rad(target(11));
dphi = deg2rad(target(12));
xdot = target(13);
ydot = target(14);
zdot = target(15);

T-G-cent(1,:)=[x y z V theta phi dx dy dz dV dtheta dphi xdot ydot zdot];

```

```

%Process Noise

TS_V(1)=target(16);
TS_V(2)=target(17);
TS_V(3)=target(18);

%Preallocation

T_C_cent=zeros(T,12);
T_G_Rotdata=zeros(T,3);
T_RotG2C=zeros(3,3,T);
T_RotC2G=zeros(3,3,T);
T_P_cent=zeros(T,5);

%% Measurement conditions

%Initial point locations (for a cube), based of T centroid
O_G_point_a=[T_G_cent(1)-.5 T_G_cent(2)-.5 T_G_cent(3)-.5];
O_G_point_b=[T_G_cent(1)+.5 T_G_cent(2)-.5 T_G_cent(3)-.5];
O_G_point_c=[T_G_cent(1)-.5 T_G_cent(2)+.5 T_G_cent(3)-.5];
O_G_point_d=[T_G_cent(1)-.5 T_G_cent(2)-.5 T_G_cent(3)+.5];
O_G_point_e=[T_G_cent(1)-.5 T_G_cent(2)+.5 T_G_cent(3)+.5];
O_G_point_f=[T_G_cent(1)+.5 T_G_cent(2)-.5 T_G_cent(3)+.5];
O_G_point_g=[T_G_cent(1)+.5 T_G_cent(2)+.5 T_G_cent(3)-.5];
O_G_point_h=[T_G_cent(1)+.5 T_G_cent(2)+.5 T_G_cent(3)+.5];

%Form the initial cube based of points

O_G_points=[O_G_point_a;O_G_point_b;O_G_point_f;O_G_point_h;...
    O_G_point_g;O_G_point_c;O_G_point_a;O_G_point_d;...
    O_G_point_e;O_G_point_h;O_G_point_f;O_G_point_d;...
    O_G_point_e;O_G_point_c;O_G_point_g;O_G_point_b];

%Number of points

O_num=16;

```

```

%Preallocation
O_G_cent=zeros(T,6);
O_C_cent=zeros(T,6);
O_RotG2C=zeros(3,3,T);
O_C_points=zeros(O_num,3,T);
O_P_points=zeros(O_num,2,T);
O_P_cent=zeros(T,5);

%Measurement noise covariances
O_V_M(1)    =    obs(1);
O_V_M(2)    =    obs(2);
O_V_M(3)    =    obs(3);

%Match initial conditions
O_G_cent(1,:)=[T_G_cent(1,[1,2,3]), T_G_cent(1,[1,2,3])];

%% Particle A Parameters
%Initial conditions
xp1    =    filter1(1);
yp1    =    filter1(2);
zp1    =    filter1(3);
Vp1    =    filter1(4);
thetap1    =    deg2rad(filter1(5));
phip1    =    deg2rad(filter1(6));

dxp1    =    filter1(7);
dyp1    =    filter1(8);
dzp1    =    filter1(9);
dVp1    =    filter1(10);
dthetap1    =    deg2rad(filter1(11));
dphip1    =    deg2rad(filter1(12));

```

```

xdotp1 = Vp1*sin(thetap1)*cos(hip1)*dt;
ydotp1 = Vp1*cos(thetap1)*dt;
zdotp1 = Vp1*sin(thetap1)*sin(hip1)*dt;

P_G_S(1,:)=[xp1 yp1 zp1 Vp1 thetap1 hip1 dxp1 dyp1 dzp1 dVp1 dthetap1...
            dhip1 xdotp1 ydotp1 zdotp1];

%Number of particles
P_num = filter1(13);

%Variances
P_V_S(1) = filter1(14); %x variance
P_V_S(2) = filter1(15); %y variance
P_V_S(3) = filter1(16); %z variance
P_V_S(4) = filter1(17); %V variance
P_V_S(5) = filter1(18); %theta variance
P_V_S(6) = filter1(19); %phi varaince

P_V_S(7) = filter1(20); %dyp
P_V_S(8) = filter1(21); %dyp
P_V_S(9) = filter1(22); %dzp
P_V_S(10) = filter1(23); %dVp
P_V_S(11) = filter1(24); %dthetap
P_V_S(12) = filter1(25); %dhip

P_V_M = filter1(26); %Measurement noise covariance

%Weighing matrix
W_u = filter1(27); %u observation weight
W_v = filter1(28); %v observation weight
W_w = filter1(29); %w observation weight

```

```

P_W = [W_u 0 0; 0 W_v 0; 0 0 W_w];

%Initial particles for global state
P_G_S_part=zeros(P_num,15,T); %Preallocation of particle matrix
for i2=1:P_num
    P_G_S_part(i2,:,1)=[P_G_S(1,1)+sqrt(P_V_S(1))*randn;...
        P_G_S(1,2)+sqrt(P_V_S(2))*randn;...
        P_G_S(1,3)+sqrt(P_V_S(3))*randn;...
        P_G_S(1,4)+sqrt(P_V_S(4))*randn;...
        P_G_S(1,5)+sqrt(P_V_S(5))*randn;...
        P_G_S(1,6)+sqrt(P_V_S(6))*randn;...
        P_G_S(1,7)+sqrt(P_V_S(7))*randn;...
        P_G_S(1,8)+sqrt(P_V_S(8))*randn;...
        P_G_S(1,9)+sqrt(P_V_S(9))*randn;...
        P_G_S(1,10)+sqrt(P_V_S(10))*randn;...
        P_G_S(1,11)+sqrt(P_V_S(11))*randn;...
        P_G_S(1,12)+sqrt(P_V_S(12))*randn;...
        P_G_S_part(i2,10,1);...
        P_G_S_part(i2,11,1);...
        P_G_S_part(i2,12,1)];
end

%Preallocation
P_G_S_part_u=zeros(P_num,15,T);
P_RotG2C=zeros(3,3,T);
P_C_O_part=zeros(P_num,3,T);
P_P_O_diff=zeros(P_num,3,T);
P_P_W=zeros(T,P_num);
P_P_O_rawW=zeros(T,P_num);
P_P_get=zeros(T,P_num);

```

```

%% Particle B conditions (Up,Vp,Wp)
%Initial conditions
xp2      =   filter2(1);
yp2      =   filter2(2);
zp2      =   filter2(3);
Vp2      =   filter2(4);
thetap2  =   deg2rad(filter2(5));
phip2    =   deg2rad(filter2(6));

dxp2     =   filter2(7);
dyp2     =   filter2(8);
dzp2     =   filter2(9);
dVp2     =   filter2(10);
dthetap2 =   deg2rad(filter2(11));
dphip2   =   deg2rad(filter2(12));

xdotp2   =   Vp2*sin(thetap2)*cos(phip2)*dt;
ydotp2   =   Vp2*cos(thetap2)*dt;
zdotp2   =   Vp2*sin(thetap2)*sin(phip2)*dt;

P2_G_S(1,:)=[xp2 yp2 zp2 Vp2 thetap2 phip2 dxp2 dyp2 dzp2 dVp2 dthetap2...
             dphip2 xdotp2 ydotp2 zdotp2];

%Number of particles
P2_num    =   filter2(13);

%Variances
P2_V_S(1) =   filter2(14); %x variance
P2_V_S(2) =   filter2(15); %y variance
P2_V_S(3) =   filter2(16); %z variance
P2_V_S(4) =   filter2(17); %V variance
P2_V_S(5) =   filter2(18); %theta variance

```

```

P2_V_S(6)      =   filter2(19); %phi varaince

P2_V_S(7)      =   filter2(20); %dxp
P2_V_S(8)      =   filter2(21); %dyp
P2_V_S(9)      =   filter2(22); %dzp
P2_V_S(10)     =   filter2(23); %dVp
P2_V_S(11)     =   filter2(24); %dthetap
P2_V_S(12)     =   filter2(25); %dphip

P2_V_M   =   filter2(26); %Measurement noise covariance

%Weight matrix
W_u      =   filter2(27); %u observation weight
W_v      =   filter2(28); %v observation weight
W_s      =   filter2(29); %s observation weight
W_udot   =   filter2(30); %wdot observation weight
W_vdot   =   filter2(31);

P2_W = [W_u 0 0 0 0; 0 W_v 0 0 0; 0 0 W_s 0 0; 0 0 0 W_udot 0;...
        0 0 0 0 W_vdot];
P2_W=[10 0 0 0 0;...
      0 10 0 0 0;...
      0 0 3 0 0;...
      0 0 0 4 0;...
      0 0 0 0 4];
P2_W=diag([10*ones(1,5)]);

%Initial particles for global state
P2_G_S_part=zeros(P2_num,15,T); %Preallocation of particle matrix
for i2=1:P2_num
    P2_G_S_part(i2,:,1)=[P2_G_S(1,1)+sqrt(P2_V_S(1))*randn;...
        P2_G_S(1,2)+sqrt(P2_V_S(2))*randn;...

```

```

P2_G_S(1,3)+sqrt(P2_V_S(3))*randn;...
P2_G_S(1,4)+sqrt(P2_V_S(4))*randn;...
P2_G_S(1,5)+sqrt(P2_V_S(5))*randn;...
P2_G_S(1,6)+sqrt(P2_V_S(6))*randn;...
P2_G_S(1,7)+sqrt(P2_V_S(7))*randn;...
P2_G_S(1,8)+sqrt(P2_V_S(8))*randn;...
P2_G_S(1,9)+sqrt(P2_V_S(9))*randn;...
P2_G_S(1,10)+sqrt(P2_V_S(10))*randn;...
P2_G_S(1,11)+sqrt(P2_V_S(11))*randn;...
P2_G_S(1,12)+sqrt(P2_V_S(12))*randn;...
P2_G_S_part(i2,10,1);...
P2_G_S_part(i2,11,1);...
P2_G_S_part(i2,12,1)];

end

%Preallocation
P2_G_S_part_u=zeros(P2_num,15,T);
P2_RotG2C=zeros(3,3,T);
P2_C_O_part=zeros(P2_num,6,T);
P2_P_O_part=zeros(P2_num,5,T);
P2_P_O_diff=zeros(P2_num,5,T);
P2_P_W=zeros(T,P2_num);

%% SLMA
L1_G_cent=zeros(T,9);
L1_G_cent(1,[1,2,3]) = O_G_cent(1,[1,2,3]);
L1_G_cent(1,[4,5,6]) = T_G_cent(1,[13,14,15]);
L1_G_cent(1,[7,8,9]) = T_G_cent(1,[4,5,6]);
L1_C_cent(1,[4,5,6]) = T_G_cent(1,[13,14,15]);
L1_G_cent(1,:)=T_G_cent(1,[1,2,3,13,14,15,4,5,6]);

%% SLMB
L2_G_cent=zeros(T,9);

```

```

L2_G_cent(1,[1,2,3])      =    O_G_cent(1,[1,2,3]);
L2_G_cent(1,[4,5,6])      =    T_G_cent(1,[13,14,15]);
L2_G_cent(1,[7,8,9])      =    T_G_cent(1,[4,5,6]);
L2_C_cent(1,[4,5,6])      =    T_G_cent(1,[13,14,15]);
L2_G_cent(1,:)=T_G_cent(1,[1,2,3,13,14,15,4,5,6]);
L2_P_cent    =    zeros(T,4);

for i=2:T %i=1 is initial conditions
%% Truth Centroid
%Update movement of centroid in global
    %Update truth centroid position
    %Position values will NOT be future values: the x used here is based
    %off of the previous x plus the previous delta_x. Thus, delta and dot
    %values pertain to the x in the same time step NOT a future x
    x=T_G_cent(i-1,1)+T_G_cent(i-1,7);
    y=T_G_cent(i-1,2)+T_G_cent(i-1,8);
    z=T_G_cent(i-1,3)+T_G_cent(i-1,9);
    V=T_G_cent(i-1,4)+T_G_cent(i-1,10);
    theta=T_G_cent(i-1,5)+T_G_cent(i-1,11);
    phi=T_G_cent(i-1,6)+T_G_cent(i-1,12);

    xdot=V*sin(theta)*cos(phi)*dt;
    ydot=V*cos(theta)*dt;
    zdot=V*sin(theta)*sin(phi)*dt;

    V_dot=T_G_cent(i-1,10)/dt;
    theta_dot=T_G_cent(i-1,11)/dt;
    phi_dot=T_G_cent(i-1,12)/dt;

    dx=V*sin(theta)*cos(phi)*dt+...
        V_dot*sin(theta)*cos(phi)*(dt^2/2)+...
        theta_dot*V*cos(theta)*cos(phi)*(dt^2/2);

```

```

dy=V*cos(theta)*dt+...
    V_dot*cos(theta)*(dt^2/2)-...
    theta_dot*V*sin(theta)*(dt^2/2);

dz=V*sin(theta)*sin(phi)*dt+...
    V_dot*sin(theta)*sin(phi)*(dt^2/2)+...
    phi_dot*V*sin(theta)*cos(phi)*(dt^2/2);

dV=V_dot*dt;
dtheta=theta_dot*dt;
dphi=phi_dot*dt;

%Add Process Noise
V=V+sqrt(TS_V(1))*randn;
theta=theta+sqrt(TS_V(2))*randn;
phi=phi+sqrt(TS_V(3))*randn;

%New centroid positions
T_G_cent(i,1)=x;
T_G_cent(i,2)=y;
T_G_cent(i,3)=z;
T_G_cent(i,4)=V;
T_G_cent(i,5)=theta;
T_G_cent(i,6)=phi;
T_G_cent(i,7)=dx;
T_G_cent(i,8)=dy;
T_G_cent(i,9)=dz;
T_G_cent(i,10)=dV;
T_G_cent(i,11)=dtheta;
T_G_cent(i,12)=dphi;
T_G_cent(i,13)=xdot;

```

```

T_G_cent(i,14)=ydot;
T_G_cent(i,15)=zdot;

%Rotate to camera orientation
%States in the camera orientation are u,v,w and dots
%Generate angles and rotation DCM
%Camera angles assume the camera tracks the truth centroid perfectly
T_G_Rotdata(i,1)=sqrt(T_G_cent(i,1)^2+T_G_cent(i,2)^2+T_G_cent(i,3)^2);
%Pan (phi)
T_G_Rotdata(i,2)=atan2(T_G_cent(i,3),T_G_cent(i,1));
%Tilt, note: tilts from y axis (positive down), so 0 is on y-axis
%(theta)
T_G_Rotdata(i,3)=acos(T_G_cent(i,2)/T_G_Rotdata(i,1));

%Calculate rate of rotation (phi dot), x
T_G_Rotdata(i,4)=(T_G_Rotdata(i,2)-T_G_Rotdata(i-1,2))/dt;

%Calculate rate of tilt (theta dot), y
T_G_Rotdata(i,5)=(T_G_Rotdata(i,3)-T_G_Rotdata(i-1,3))/dt;

%Create DCM based on current rotation angles
T_DCM_G2C=@(angz) Rot_y(angz(1)-pi/2)*Rot_x(angz(2)-pi/2)*Rot_z(pi/2);
T_RotG2C(:, :, i)=T_DCM_G2C(T_G_Rotdata(i, [2,3])).';

%Rotate centroid to camera frame
T_C_cent(i, [1,2,3])=T_RotG2C(:, :, i)*T_G_cent(i, [1,2,3]); %u,v,w
T_C_cent(i, [4,5,6])=T_RotG2C(:, :, i)*T_G_cent(i, [13,14,15]); %dot u v w

%Convert to pixel values
T_P_cent(i,1)=(T_C_cent(i,1)/T_C_cent(i,3))*f; %u_p
T_P_cent(i,2)=(T_C_cent(i,2)/T_C_cent(i,3))*f; %v_p
T_P_cent(i,3)=(T_C_cent(i,6)/T_C_cent(i,3))*f; %w

```

```

T_P_cent(i,4)=(T_C_cent(i,4)/T_C_cent(i,3))*f; %u_pdot
T_P_cent(i,5)=(T_C_cent(i,5)/T_C_cent(i,3))*f; %v_pdot

%% Observation Centroid and Points
%Update global point position (CUBE)
for i2=1:O_num
    O_G_points(i2,1,i)=O_G_points(i2,1,i-1)+T_G_cent(i-1,7);
    O_G_points(i2,2,i)=O_G_points(i2,2,i-1)+T_G_cent(i-1,8);
    O_G_points(i2,3,i)=O_G_points(i2,3,i-1)+T_G_cent(i-1,9);
end

%Calculate observed centroid based on mean of points
O_G_cent(i,1)=mean(O_G_points(:,1,i));
O_G_cent(i,2)=mean(O_G_points(:,2,i));
O_G_cent(i,3)=mean(O_G_points(:,3,i));

%Rotate to camera orientation (use same cam rot angles as in truth)
%Rotate points to camera orientation
O_RotG2C(:, :, i)=T_RotG2C(:, :, i);
O_noise=[sqrt(O_V_M(1))*randn; sqrt(O_V_M(2))*randn;...
    sqrt(O_V_M(3))*randn];
for i2=1:O_num
    O_C_points(i2, :, i)=O_RotG2C(:, :, i)*O_G_points(i2, :, i)'+O_noise;
end

%Calculate observed centroid (in camera frame)
O_C_cent(i,1)=mean(O_C_points(:,1,i));
O_C_cent(i,2)=mean(O_C_points(:,2,i));
O_C_cent(i,3)=mean(O_C_points(:,3,i));

%Calculate u,v,w velocities (difference between the two positions)

```

```

%Assume all points move with the same velocities, so we use the truth
%velocities, u v w dot
O_C_cent(i,[4,5,6])= T_C_cent(i,[4,5,6]);
T_RotC2G(:, :, i)=transpose(T_RotG2C(:, :, i));
O_G_cent(i,[4,5,6])=T_RotC2G(:, :, i)*O_C_cent(i,[1,2,3])';

%Calculate and convert to pixel values
%Calculate u and v pixel values for each point
for i2=1:O_num
    O_P_points(i2,1,i)=(O_C_points(i2,1,i)/O_C_points(i2,3,i))*f; %u_p
    O_P_points(i2,2,i)=(O_C_points(i2,2,i)/O_C_points(i2,3,i))*f; %v_p
end

%Calculate s, u_dot, v_dot using least mean squares method
state=[O_P_points(:,1,i-1)' O_P_points(:,2,i-1)'];...
    zeros(1,O_num) dt*ones(1,O_num);...
    dt*ones(1,O_num) zeros(1,O_num)];

b=[O_P_points(:,1,i)' O_P_points(:,2,i)'];

%    A=b*state'*pinv(state*state');
A=b*(state'*state)^-1*state'*state;

%Create pixel 'centroid'
O_P_cent(i,1)=mean(O_P_points(:,1,i)); %u_p
O_P_cent(i,2)=mean(O_P_points(:,2,i)); %v_p
O_P_cent(i,3)=A(1,1); %s
O_P_cent(i,4)=A(1,2); %u_p_dot
O_P_cent(i,5)=A(1,3); %v_p_dot

%% Particle Centroid and Points (U,V,W)

```

```

%Update particles (state and observations)
%As with the Truth cent, take in the previous time step values
for i2=1:P_num
    %Update state
    x=P_G_S_part(i2,1,i-1)+P_G_S_part(i2,7,i-1);
    y=P_G_S_part(i2,2,i-1)+P_G_S_part(i2,8,i-1);
    z=P_G_S_part(i2,3,i-1)+P_G_S_part(i2,9,i-1);

    V=P_G_S_part(i2,4,i-1)+P_G_S_part(i2,10,i-1)+sqrt(P_V_S(4))*randn;
    theta=P_G_S_part(i2,5,i-1)+P_G_S_part(i2,11,i-1)+...
        sqrt(P_V_S(5))*randn;
    phi=P_G_S_part(i2,6,i-1)+P_G_S_part(i2,12,i-1)+...
        sqrt(P_V_S(6))*randn;

    %Angle adjustment: ensure V is not negative, angles in proper range
    if(V<0)
        V=abs(V);
    end

    %Fix and reduce angles
    theta=wrapTo2Pi(theta);
    if(theta>pi())
        theta_a=theta-pi();
        theta=pi()-theta_a;
    end

    xdot=V*sin(theta)*cos(phi)*dt;
    ydot=V*cos(theta)*dt;
    zdot=V*sin(theta)*sin(phi)*dt;

    V_dot=P_G_S_part(i2,10,i)/dt;
    theta_dot=P_G_S_part(i2,11,i)/dt;

```

```

phi_dot=P_G_S_part (i2,12,i)/dt;

dx=V*sin(theta)*cos(phi)*dt+...
    V_dot*sin(theta)*cos(phi)*(dt^2/2)+...
    theta_dot*V*cos(theta)*cos(phi)*(dt^2/2)+...
    -phi_dot*V*sin(theta)*sin(phi)*(dt^2/2);

dy=V*cos(theta)*dt+...
    V_dot*cos(theta)*(dt^2/2)-...
    theta_dot*V*sin(theta)*(dt^2/2);

dz=V*sin(theta)*sin(phi)*dt+...
    V_dot*sin(theta)*sin(phi)*(dt^2/2)+...
    theta_dot*V*cos(theta)*sin(phi)*(dt^2/2)+...
    phi_dot*V*sin(theta)*cos(phi)*(dt^2/2);

dV=V_dot*dt;
dtheta=theta_dot*dt;
dphi=phi_dot*dt;

%New centroid positions
P_G_S_part_u(i2,1,i)=x;
P_G_S_part_u(i2,2,i)=y;
P_G_S_part_u(i2,3,i)=z;
P_G_S_part_u(i2,4,i)=V;
P_G_S_part_u(i2,5,i)=theta;
P_G_S_part_u(i2,6,i)=phi;
P_G_S_part_u(i2,7,i)=dx;
P_G_S_part_u(i2,8,i)=dy;
P_G_S_part_u(i2,9,i)=dz;
P_G_S_part_u(i2,10,i)=dV;
P_G_S_part_u(i2,11,i)=dtheta;

```

```

P_G_S_part_u(i2,12,i)=dphi;
P_G_S_part_u(i2,13,i)=xdot;
P_G_S_part_u(i2,14,i)=ydot;
P_G_S_part_u(i2,15,i)=zdot;

%Observation Update (ie. what we think the camera will see based on
%the states)
%Rotate to camera frame (use same angles as in Truth)
P_RotG2C(:, :, i)=T_RotG2C(:, :, i);
P_C_O_part(i2, [1,2,3], i)=P_RotG2C(:, :, i)*...
    P_G_S_part_u(i2, [1,2,3], i)';

%Calculate difference between observation (measurement) and filter
%prediction
P_P_O_diff(i2, :, i)=O_C_cent(i, [1,2,3])-P_C_O_part(i2, :, i);

%Weights to be used
P_P_O_rawW(i, i2)=P_P_O_diff(i2, :, i)*P_W*P_P_O_diff(i2, :, i)';

%Weight particles
P_P_W(i, i2)=(1/sqrt(2*pi*P_V_M))*exp(-(P_P_O_diff(i2, :, i)*P_W*...
    P_P_O_diff(i2, :, i)')/(2*P_V_M));
end

%Normalize to form a probability distribution (ie. sums to 1)
P_P_W(i, :)=P_P_W(i, :)/sum(P_P_W(i, :));

%Resampling: from this new distribution, we randomly resample from it
%to generate new estimate particles
for i2=1:P_num
    P_P_get(i, i2)=find(rand<=cumsum(P_P_W(i, :)), 1);
    P_G_S_part(i2, :, i)=P_G_S_part_u(P_P_get(i, i2), :, i);
end

```

```

end

%The final estimate, state, is a metric of the final resampling
P_G_S(i,:)=mean(P_G_S_part(:, :, i));

%% Particle Centroid and Points (Up,Vp,Wp)
%Select model variation/parameter
%Noise on vector:    1
%Jacobain:           2
%No correction:      0

%Update particles (state and observations)
%As with the Truth cent, take in the previous time step values
switch correct
case 1 %Noise on vector
    temp_r_vec=zeros(1,3);
    [temp_r_vec(1),temp_r_vec(2),temp_r_vec(3)] = sph2cart(...
        T_G_Rotdata(i,2),T_G_Rotdata(i,3),T_G_Rotdata(i,1));
    temp_r_vec=temp_r_vec/norm(temp_r_vec)*7*randn;
%    temp_r_vec = T_RotG2C(end, :, i)*7*randn;
    temp_r_vec = T_RotG2C(end, :, i);
    temp_randVtp=[.1*sqrt(P2_V_S(4)) 0 0; 0 .1*sqrt(P2_V_S(5)) 0;...
        0 0 .1*sqrt(P2_V_S(6))];

case 2
    temp_r_vec=zeros(1,3); % Prevents any noise being introduced
    if i>3
        %Retrieve previous estimated global state values, reassign to variables
        %for ease of calculation and jacobian generation
        V      = P2_G_S(i-1,4);
        theta  = P2_G_S(i-1,5);
        phi    = P2_G_S(i-1,6);

```

```

%First Jacobian, uses V, theta, phi, and returns dx, dy, dz
V_Jacob1(:, :, i) = [cos(theta)*sin(phi)*dt, -V*sin(theta)*...
    sin(phi)*dt, V*cos(theta)*cos(phi)*dt; ...
    cos(phi)*dt, 0, -V*sin(phi)*dt; ...
    sin(theta)*sin(phi)*dt, V*cos(theta)*sin(phi)*dt, ...
    V*cos(phi)*sin(theta)*dt];

%Second Jacobian
w = P2_C_S(i-1, 3);
V_Jacob2(:, :, i) = [0, 0, 1/w; 1/w, 0, 0; 0, 1/w, 0];

%Generate full matrix
V_JacobAll(:, :, i) = V_Jacob2(:, :, i)*P2_RotG2C(:, :, i-1)*...
    V_Jacob1(:, :, i);

%Generate variance values, based on eigen values that
%define the amount of correlation between state and
%variable and the vectors that indicate if this
%correlation is increasing
[temp_Evec, temp_Eval] = eig(V_JacobAll(:, :, i)).'*...
    V_JacobAll(:, :, i));
temp_randVtp = 10*temp_Evec*diag(4e-5*(diag(temp_Eval)+...
    4e-6).^(-1));

else

    temp_Evec(:, 1) = [1; 0; 0];
    temp_Eval(1, 1) = 1e-4;
    temp_Evec(:, 2) = [0; 1; 0];
    temp_Eval(2, 2) = 1e-2;
    temp_Evec(:, 3) = [0; 0; 1];
    temp_Eval(3, 3) = 1e-2;
    temp_randVtp = 3*temp_Evec*diag(4e-5*(diag(temp_Eval)+4e-6).^(-1));

end

case 0
    temp_r_vec = zeros(1, 3);

```

```

temp_randVtp=[.1*sqrt(P2_V_S(4)) 0 0; 0 .1*sqrt(P2_V_S(5)) 0;...
0 0 .1*sqrt(P2_V_S(6))];

end

for i2=1:P2_num

    %Generate variances
    temp_r_vec=temp_r_vec.*[sqrt(P2_V_S(1))*randn,...
        sqrt(P2_V_S(2))*randn, sqrt(P2_V_S(3))*randn];
    temp_randVtp2=temp_randVtp*randn(3,1);

    %Update state
    x=    P2_G_S_part(i2,1,i-1)+P2_G_S_part(i2, 7,i-1)+temp_r_vec(1);
    y=    P2_G_S_part(i2,2,i-1)+P2_G_S_part(i2, 8,i-1)+temp_r_vec(2);
    z=    P2_G_S_part(i2,3,i-1)+P2_G_S_part(i2, 9,i-1)+temp_r_vec(3);

    V=    P2_G_S_part(i2,4,i-1)+P2_G_S_part(i2,10,i-1)+...
        temp_randVtp2(1);
    theta=P2_G_S_part(i2,5,i-1)+P2_G_S_part(i2,11,i-1)+...
        temp_randVtp2(2);
    phi=   P2_G_S_part(i2,6,i-1)+P2_G_S_part(i2,12,i-1)+...
        temp_randVtp2(3);

    %Angle adjustment: ensure V is not negative, angles in proper range
    if (V<0)
        V=abs(V);
    end

    xdot=V*sin(theta)*cos(phi)*dt;
    ydot=V*cos(phi)*dt;
    zdot=V*sin(theta)*sin(phi)*dt;

```

```

V_dot=P2_G_S_part (i2,10,i-1)/dt;
theta_dot=P2_G_S_part (i2,11,i-1)/dt;
phi_dot=P2_G_S_part (i2,12,i-1)/dt;

dx=V*sin(theta)*cos(phi)*dt+...
    V_dot*sin(theta)*cos(phi)*(dt^2/2)+...
    theta_dot*V*cos(theta)*cos(phi)*(dt^2/2)+...
    -phi_dot*V*sin(theta)*sin(phi)*(dt^2/2);

dy=V*cos(theta)*dt+...
    V_dot*cos(theta)*(dt^2/2)-...
    theta_dot*V*sin(theta)*(dt^2/2);

dz=V*sin(theta)*sin(phi)*dt+...
    V_dot*sin(theta)*sin(phi)*(dt^2/2)+...
    theta_dot*V*cos(theta)*sin(phi)*(dt^2/2)+...
    phi_dot*V*sin(theta)*cos(phi)*(dt^2/2);

dV=V_dot*dt;
dtheta=theta_dot*dt;
dphi=phi_dot*dt;

%New centroid positions
P2_G_S_part_u (i2,1,i)=x;
P2_G_S_part_u (i2,2,i)=y;
P2_G_S_part_u (i2,3,i)=z;
P2_G_S_part_u (i2,4,i)=V;
P2_G_S_part_u (i2,5,i)=theta;
P2_G_S_part_u (i2,6,i)=phi;
P2_G_S_part_u (i2,7,i)=dx;
P2_G_S_part_u (i2,8,i)=dy;
P2_G_S_part_u (i2,9,i)=dz;

```

```

P2_G_S_part_u(i2,10,i)=dV;
P2_G_S_part_u(i2,11,i)=dtheta;
P2_G_S_part_u(i2,12,i)=dphi;
P2_G_S_part_u(i2,13,i)=xdot;
P2_G_S_part_u(i2,14,i)=ydot;
P2_G_S_part_u(i2,15,i)=zdot;

%Observation Update (ie. what we think the camera will see based on
%the states)

%Rotate to camera frame (use same angles as in Truth)
P2_RotG2C(:, :, i)=T_RotG2C(:, :, i);
P2_C_O_part(i2, [1,2,3], i)=P2_RotG2C(:, :, i)*...
    P2_G_S_part_u(i2, [1,2,3], i)';
P2_C_O_part(i2, [4,5,6], i)=P2_RotG2C(:, :, i)*...
    P2_G_S_part_u(i2, [13,14,15], i)';

%Obtain pixel values, u,v,s,udot,vdot, based on camera states
P2_P_O_part(i2,1,i)=P2_C_O_part(i2,1,i)/P2_C_O_part(i2,3,i)*f; %u_p
P2_P_O_part(i2,2,i)=P2_C_O_part(i2,2,i)/P2_C_O_part(i2,3,i)*f; %v_p
P2_P_O_part(i2,3,i)=P2_C_O_part(i2,6,i)/P2_C_O_part(i2,3,i)*f; %new s
%
%   P2_P_O_part(i2,3,i)=(-(P2_C_O_part(i2,1,i)*...
%
%       P2_C_O_part(i2,6,i))/P2_C_O_part(i2,3,i)^2)+...
%
%       -(P2_C_O_part(i2,2,i)*P2_C_O_part(i2,6,i))/...
%
%       P2_C_O_part(i2,3,i)^2))/2; %s_p
P2_P_O_part(i2,4,i)=(P2_C_O_part(i2,4,i)/P2_C_O_part(i2,3,i))*f;
P2_P_O_part(i2,5,i)=(P2_C_O_part(i2,5,i)/P2_C_O_part(i2,3,i))*f;

%Calculate difference between observed centroid and particle cent
P2_P_O_diff(i2, :, i)=O_P_cent(i, :)-P2_P_O_part(i2, :, i);
P2_P_O_diff(i2, :, i)=T_P_cent(i, :)-P2_P_O_part(i2, :, i);

%Weights to be used

```

```

P2_P_O_rawW(i,i2)=P2_P_O_diff(i2,:,i)*P2_W*P2_P_O_diff(i2,:,i)';

%Weight particles
P2_P_W(i,i2)=(1/sqrt(2*pi*P2_V_M))*exp(-(P2_P_O_diff(i2,:,i)*...
P2_W*P2_P_O_diff(i2,:,i)')/(2*P2_V_M));
end

stop=mean(sum(P2_P_W(i,:)));
if stop < 1e-175
    stop;
    flag=1;
    break
end

P2_P_O_diff(i2,:,i)

%% Plot distribution of weighted particles in real time
%% for diagnostic purposes
figure(72);clf;
set(gcf,'position',[ 680 72 1152 906]);
a=subplot(3,2,1);
stem3(P2_P_O_part(:,1,i),P2_P_O_part(:,2,i),P2_P_W(i,:), 'ro');
hold on; stem3(T_P_cent(i,1),T_P_cent(i,2),max(P2_P_W(i,:)), 'k*');
title(a, 'U vs V Position Particle Weight');
%    daspect([1e1,1e1,1e-2]);drawnow;
a=subplot(3,2,3);
stem(P2_P_O_part(:,3,i),P2_P_W(i,:), 'ro');
hold on; stem(T_P_cent(i,3),max(P2_P_W(i,:)), 'k*');
title(a, 'Z Position Particle Weight');
%    daspect([1e-4,1e-2,1]);drawnow;
a=subplot(3,2,5);
stem3(P2_P_O_part(:,4,i),P2_P_O_part(:,5,i),P2_P_W(i,:), 'ro');

```

```

hold on; stem3(T_P_cent(i,4),T_P_cent(i,5),max(P2_P_W(i,:), 'k*'));
title(a, 'U vs V Velocity Particle Weight');
%   daspect([1e2,1e2,1e-2]);drawnow;

a=subplot(3,2,2);
stem3(P2_G_S_part_u(:,1,i),P2_G_S_part_u(:,2,i),P2_P_W(i,:), 'ro');
hold on; stem3(T_G_cent(i,1),T_G_cent(i,2),max(P2_P_W(i,:), 'k*'));
title(a, 'X vs Y Position Particle Weight');
%   daspect([1e1,1e1,1e-2]);drawnow;
a=subplot(3,2,4);
stem(P2_G_S_part_u(:,3,i),P2_P_W(i,:), 'ro');
hold on; stem(T_G_cent(i,3),max(P2_P_W(i,:), 'k*'));
title(a, 'Z Position Particle Weight');
%   daspect([1e1,1e1,1e-2]);drawnow;
a=subplot(3,2,6);
stem3(ang_red(P2_G_S_part_u(:,5,i)),ang_red(P2_G_S_part_u(:,6,i)),...
      P2_P_W(i,:), 'ro');
hold on; stem3(ang_red(T_G_cent(i,3)),ang_red(T_G_cent(i,4)),...
      max(P2_P_W(i,:), 'k*'));
title(a, 'X vs Y Velocity Particle Weight');
%   daspect([1e1,1e1,1e-2]);drawnow;
pause();

%%
if (flag==1)
    break
end
% P2_P_O_diffmean(i,:)=mean(P2_P_O_diff(:, :, i));
%Normalize to form a probability distribution (ie. sums to 1)
P2_P_W(i,:)=P2_P_W(i,:)./sum(P2_P_W(i,:));

%Resampling: from this new distribution, we randomly resample from it

```

```

%to generate new estimate particles

for i2=1:P2_num
    P2_P_get(i,i2)=find(rand<=cumsum(P2_P_W(i,:)),1);
    P2_G_S_part(i2,:,i)=P2_G_S_part_u(P2_P_get(i,i2),:,i);
end

%The final estimate, state, is a metric of the final resampling
P2_G_S(i,:)=mean(P2_G_S_part(:, :, i));

%Provide P2_C_S for variance calculations
P2_C_S(i,[1,2,3])=P2_RotG2C(:, :, i)*P2_G_S(i,[4,5,6])';
P2_C_S(i,[4,5,6])=P2_RotG2C(:, :, i)*P2_G_S(i,[10,11,12])';

%% SLMA

L1_C_cent(i,1)    =    O_C_cent(i-1,1)+L1_C_cent(i-1,4)*dt;
L1_C_cent(i,2)    =    O_C_cent(i-1,2)+L1_C_cent(i-1,5)*dt;
L1_C_cent(i,3)    =    O_C_cent(i-1,3)+L1_C_cent(i-1,6)*dt;

%Velocities
L1_C_cent(i,4)    =    (O_C_cent(i,1)-O_C_cent(i-1,1))/dt;
L1_C_cent(i,5)    =    (O_C_cent(i,2)-O_C_cent(i-1,2))/dt;
L1_C_cent(i,6)    =    (O_C_cent(i,3)-O_C_cent(i-1,3))/dt;

%Rotate to global from camera (inverse of DCM)
L1_G_cent(i,[1,2,3])=T_RotC2G(:, :, i)*L1_C_cent(i,[1,2,3])';

L1_G_cent(i,4)=(L1_G_cent(i,1)-L1_G_cent(i-1,1))/dt;
L1_G_cent(i,5)=(L1_G_cent(i,2)-L1_G_cent(i-1,2))/dt;
L1_G_cent(i,6)=(L1_G_cent(i,3)-L1_G_cent(i-1,3))/dt;

```

```

V    =    sqrt(L1_G_cent(i,4)^2+L1_G_cent(i,5)^2+L1_G_cent(i,6)^2);
theta    =    acos(L1_G_cent(i,5)/V);
phi      =    atan2(L1_G_cent(i,6),L1_G_cent(i,4));

L1_G_cent(i,7)    =    V;
L1_G_cent(i,8)    =    theta;
L1_G_cent(i,9)    =    phi;

%% SLMB
%Update up, vp, s (w)

%Note: normally w=s*scale factor of target, but since the obs points
%are from the edge of the target, this scale factor =1, so it is not
%explicitly stated

%Find u, w, v
L2_C_cent(i,1)    =    (O_P_cent(i,3)*O_P_cent(i,1))/f;
L2_C_cent(i,2)    =    (O_P_cent(i,3)*O_P_cent(i,2))/f;
L2_C_cent(i,3)    =    sqrt(((L2_C_cent(i,1)-L2_C_cent(i,2))*...
    L2_C_cent(i-1,6))/(2*O_P_cent(i,3)));
L2_C_cent(i,4)    =    (O_P_cent(i,3)*O_P_cent(i,4))/f;
L2_C_cent(i,5)    =    (O_P_cent(i,3)*O_P_cent(i,5))/f;
L2_C_cent(i,6)    =    (O_P_cent(i,3)-O_P_cent(i-1,3))/dt;

L2_C_cent(i,:);

%Rotate to global from camera (inverse of DCM)
L2_G_cent(i,[1,2,3])=T_RotC2G(:, :, i)*L2_C_cent(i,[1,2,3])';

L2_G_cent(i,4)=(L2_G_cent(i,1)-L2_G_cent(i-1,1))/dt;
L2_G_cent(i,5)=(L2_G_cent(i,2)-L2_G_cent(i-1,2))/dt;
L2_G_cent(i,6)=(L2_G_cent(i,3)-L2_G_cent(i-1,3))/dt;

```

```

V    =    sqrt(L2_G_cent(i,4)^2+L2_G_cent(i,5)^2+L2_G_cent(i,6)^2);
theta    =    acos(L2_G_cent(i,5)/V);
phi      =    atan2(L2_G_cent(i,6),L2_G_cent(i,4));

L2_G_cent(i,7)    =    V;
L2_G_cent(i,8)    =    theta;
L2_G_cent(i,9)    =    phi;
end

%Angle adjustment: ensure V is not negative, angles in proper range
for i=2:T
    %Terminate if EPF-B collapses
    if(flag==1)
        break
    end

    %Target
    T_G_cent(i,5)=wrapTo2Pi(T_G_cent(i,5));
    T_G_cent(i,6)=wrapTo2Pi(T_G_cent(i,6));

    if(T_G_cent(i,4)<0)
        T_G_cent(i,4)=abs(T_G_cent(i,4));
        T_G_cent(i,6)=wrapTo2Pi(T_G_cent(i,6)+pi());
        T_G_cent(i,5)=pi()-T_G_cent(i,5);
    end

    %Fix and reduce angles
    if(T_G_cent(i,5)>pi())
        theta_a=T_G_cent(i,5)-pi();
        T_G_cent(i,5)=pi()-theta_a;
        T_G_cent(i,6)=wrapTo2Pi(T_G_cent(i,6)+pi());
    end
end

```

```

%EPF-A

%Only need to adjust phi, v is already only absolute and theta has
%already been constrained
%DO NOT USE WRAPTO2PI!!!

phi=P_G_S(i,6);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
P_G_S(i,6)=abs(phi);

%EPF-B

%Only need to adjust phi, v is already only absolute and theta has
%already been constrained
%DO NOT USE WRAPTO2PI!!!

phi=P2_G_S(i,6);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

```

```

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
P2_G_S(i,6)=abs(phi);

%SLMA
%Angle adjustment: ensure V is not negative, angles in proper range
phi=L1_G_cent(i,9);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
L1_G_cent(i,9)=abs(phi);

%Fix and reduce angles
theta=L1_G_cent(i,8);
theta=wrapTo2Pi(theta);

```

```

if(theta>pi())
    theta_a=theta-pi();
    theta=pi()-theta_a;
end
L1_G_cent(i,8)=theta;

%SLMB
%Angle adjustment: ensure V is not negative, angles in proper range
phi=L2_G_cent(i,9);
if (phi > 2*pi())
    mult=floor(phi/(2*pi()));
    phi=phi-mult*2*pi();
end

if (phi < -2*pi())
    mult=floor(phi/(-2*pi()));
    phi=phi+mult*2*pi();
end

if (phi < 0)
    phi=2*pi()+phi;
end
L2_G_cent(i,9)=abs(phi);

%Fix and reduce angles
theta=L2_G_cent(i,8);
theta=wrapTo2Pi(theta);
if(theta>pi())
    theta_a=theta-pi();
    theta=pi()-theta_a;
end

```

```
L2_G_cent(i,8)=theta;  
end
```

Bibliography

- [1] “IEEE Conferences on Computer Vision and Pattern Recognition, 2009-2013”. URL <http://ieeexplore.ieee.org/servlet/opac?punumber=1000147>.
- [2] Arulampalam, M. Sanjeev, Simon Maskell, Neil Gordon, and Tim Clapp. “A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking”. *IEEE Transactions on Signal Processing*, 50:174–188, 2002.
- [3] Bayes, Thomas and Richard Price. “An Essay towards solving a Problem in the Doctrine of Chances”. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763. URL <http://www.stat.ucla.edu/history/essay.pdf>.
- [4] Bengtsson, Thomas, Peter Bickel, and Bo Li. *Control System Design: An Introduction to State-Space Methods*. Dover Publications, Inc. Mineola New York, 1986.
- [5] Bengtsson, Thomas, Peter Bickel, and Bo Li. *Probability and Statistics: Essays in Honor of David A. Freedman*. Institute of Mathematical Statistics, Beachwood Ohio, 2008.
- [6] Bimbo, Alberto De and Fabrizio Dini. “Particle Filter-Based Visual Tracking with a First Order Dynamic Model and Uncertainty Adaptation”. *Computer Vision and Image Understanding*, 115(6):771–786, June 2011.
- [7] Burl, Jeffrey R. *Linear Optimal Control: H_2 and H_∞* . Addison Wesley Longman, Inc., Menlo Park California, 1998.
- [8] Burtch, Robert C. “History of Photogrammetry”. SURE 340 Lecture Notes.
- [9] Corporation, Nikon. “Digital SLR Camera Basics”. Online Topic Notes. URL <http://imaging.nikon.com/history/basics/19/01.htm>.
- [10] Daniel D. Doyle, Alan L. Jennings and Jonathan T. Black. “Real-Time, Multiple PTZ Camera Object Tracking using Optical Flow Background Estimation”.
- [11] Doucet, Arnaud, Nando de Freitas, and Neil Gordon (editors). *Sequential Monte Carlo Methods in Practice*. Springer, 2001.
- [12] Doyle D. Doyle, Lieutenant Colonel. *Real-Time, Multiple, Pan/Tilt/Zoom, Computer Vision Tracking, and 3D Position Estimating System for Small Unmanned Aircraft System Research*. Ph.D. thesis, Air Force Institute of Technology, 2013.
- [13] Gordon, N.J., D.J. Salmond, and A.F.M. Smith. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. *Radar and Signal Processing, IEEE Proceedings F*, 140:107–113, Apr 1993.

- [14] Hammersley, J. M. and K. W. Morton. “Poor Man’s Monte Carlo”. *Journal of the Royal Statistical Society. Series B (Methodological)*, 16(1):23–38, 1954.
- [15] He, Ting, Chatschik Bisdikian, Lance Kaplan, Wei Wei, and Don Towsley. “Multi-Target Tracking Using Proximity Sensors”. Military Communications Conference, 2010 - MILCOM 2010, October-November 2010.
- [16] Li, Anping, Zhongliang Jing, and Shiqiang Hu. “Robust Observation Model for Visual Tracking in Particle Filter”. *AEU - International Journal of Electronics and Communications*, (61):186–194, 2007.
- [17] Liu, Jie, Wilson Wang, and Fai Ma. “A Regularized Auxiliary Particle Filtering Approach for System State Estimation and Battery Life Prediction”. *Smart Materials and Structures*, 20(7):075021, 2011.
- [18] Magree, Daniel P. *A Photogrammetry-Based Hybrid System For Dynamic Tracking And Measurement*. Master’s thesis, Air Force Institute of Technology, June 2010.
- [19] McElhoe, Bruce A. “An Assessment of the Navigation and Course Corrections for a Manned Flyby of Mars or Venus”. *IEEE Transactions on Aerospace and Electronics Systems*, AES-2(4):613–623, 1966.
- [20] Orderud, Fredrik. “Comparison of Kalman Filter Estimation Approaches for State Space Models with Nonlinear Measurements”, 2005. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.135.9250>.
- [21] Papoulis, Athanasios (editor). *Probability, Random Variables, and Stochastic Processes, 2nd ed.* New York: McGraw-Hill, 1984.
- [22] Smith, Gerald L., Stanley F. Schmidt, and Leonard A. McGee. “Application for Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board a Circumlunar Vehicle”, 1962.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 27-03-2014		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Oct 2012–Mar 2014		
4. TITLE AND SUBTITLE Computer Vision Tracking Using Particle Filters for 3D Position Estimation				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Kenerley, Kyle D., Second Lieutenant, USAF				5d. PROJECT NUMBER 14Y151		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB, OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENY-14-M-28		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Labs Space Vehicles Directorate 3550 Aberden Ave. Kirtland AFB, NM 87117				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RV		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED						
13. SUPPLEMENTARY NOTES This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT This line of research seeks to increase knowledge of a tracked target using the particle filter, also known as Sequential Monte Carlo (SMC) methods. The target is tracked using vision based observations. These observations were simulated using both dual cameras and a single camera. If only a single camera tracks the target, depth cannot be determined directly and is considered an unobservable state. Filters can estimate this unobservable state using a dynamic model and data from the image. However the movement of the target is nonlinear which eliminated filters traditionally used to track motion such as the Kalman filter and its variants. The particle filter is an alternative that can track nonlinear motion, but was not feasible until recently due to its computational requirements. Simulations of nonlinear target movement, first in two dimensions, then three, evaluated the particle filter's feasibility and performance. Subsequent simulations evaluated the particle filter's ability to track a target using dual and single camera observations. Evaluation tests were devised to characterize the performance of each filter. Analysis metrics were produced to analyze the results of these tests. Linear and Kalman filters were also devised to serve as additional comparisons to the particle filter. Results for dual camera observations demonstrated the filter could track the target and determine unobservable states, however results for the single camera observations indicated the filter was problematic since it could not return accurate depth estimates and suffered from severe weight collapse.						
15. SUBJECT TERMS particle filter, vision, tracking						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Alan L. Jennings (AFIT/ENY)	
U	U	U	UU	289	19b. TELEPHONE NUMBER (include area code) (937) 255-3636 x7495 alan.jennings@afit.edu	